MEMETRON: Memetic Response Optimizer for Reward-Guided Post-Decoding Optimization of Large Language Model

Anonymous authors
Paper under double-blind review

Abstract

Modern large language models (LLMs) are commonly optimized using scalar reward signals defined over completed responses, applied both during training and at inference time. However, most such reward-guided post-decoding methods remain one-shot: they independently sample a set of responses, score each once, and select the best. Staying shallow and narrow leaves higher-reward responses unrealized, while scaling up to shallow and wide sampling exacerbates reward hacking, making downstream selection methods such as best-of-N and self-consistency unreliable. We propose MEMETRON, a memetic optimization framework that formulates reward-guided post-decoding optimization (RPDO) as discrete black-box optimization over completed responses. MEMETRON alternates between GENETRON for population-based optimization and ANNETRON for annealing-based local refinement under a black-box scalar reward. Across mathematical reasoning and instruction-following tasks, MEMETRON reliably discovers higher-scoring responses. On mathematical reasoning, MEMETRON increases pass@k correctness coverage and improves the selection reliability of best-of-N and self-consistency; on instruction following, it improves LLM judge preference. On verifiable tasks, MEMETRON can incorporate ground-truth correctness via reward shaping. Comparing shaped and unshaped runs exposes extreme cases of RM-correctness misalignment, and the resulting contrastive pairs serve as training signal for reward model fine-tuning, rejection sampling SFT warmups for RL-based training pipelines such as PPO and GRPO, and direct preference learning such as DPO.

1 Introduction

Large language models (LLMs) have demonstrated strong capabilities across a wide range of generation, reasoning, and decision-making tasks. However, achieving aligned behavior requires techniques beyond pretraining, which operate at two complementary stages. The first is *post-training*, where model parameters are updated via offline methods such as SFT (Radford et al., 2018) and DPO (Rafailov et al., 2023) on fixed datasets, or via online on-policy RL methods such as PPO (Schulman et al., 2017; Ouyang et al., 2022) and GRPO (Shao et al., 2024) on self-generated rollouts, typically initialized from an SFT checkpoint. The second is *inference time*, where model parameters are fixed and outputs are guided via prompt engineering, guided decoding, or response engineering.

Offline post-training methods such as SFT with rejection sampling (Bai et al., 2022; Touvron et al., 2023; Yuan et al., 2023) and DPO with rejection sampling (Grattafiori et al., 2024), as well as inference-time response engineering methods such as best-of-N (Cobbe et al., 2021; Nakano et al., 2021), share a key pattern: the use of a reward signal, typically instantiated as a reward model (RM) that scores completed responses. Using the reward information, SFT with rejection sampling retains high-scoring candidates as fine-tuning examples, while DPO with rejection sampling builds preference pairs by contrasting high- and low-scoring samples. At inference time, best-of-N sampling selects the highest-scoring of N responses for a query.

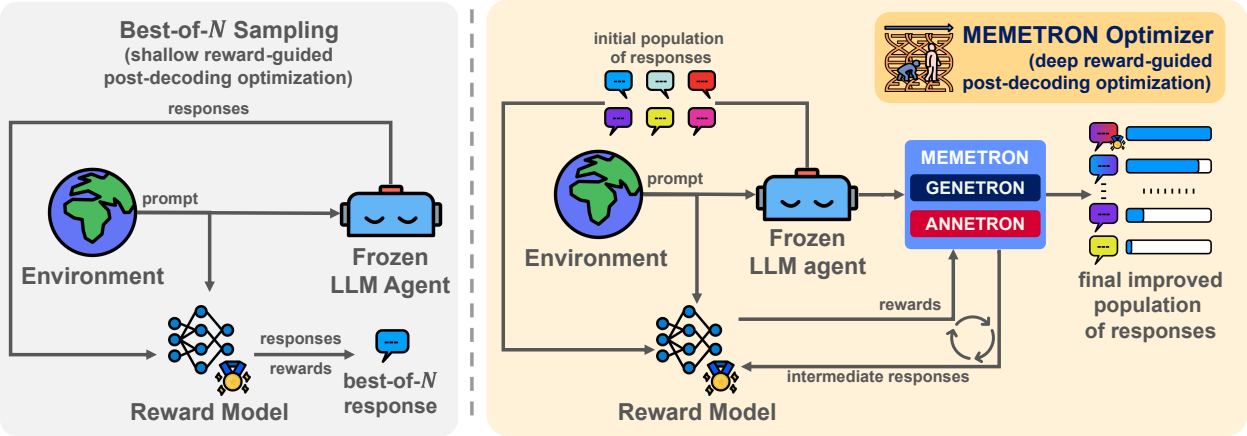

Figure 1: Comparison of shallow and deep Reward-Guided Post-Decoding Optimization (RPDO) under a scalar reward model (RM). *Shallow RPDO:* Best-of-N independently samples candidates, scores each once, and selects the highest-reward response. *Deep RPDO:* MEMETRON performs structured search over the response space, combining population-based optimization via GENETRON and annealing-based local refinement via ANNETRON, and discovers higher-reward responses unreachable by shallow search.

We call this family Reward-Guided Post-Decoding Optimization (RPDO), procedures that optimize completed responses under a scalar reward signal without updating model parameters. RPDO is reward-agnostic, applicable to any scalar signal including internal log-probabilities, programmatic verifiers, simulators, rubric-based scoring, human feedback, or composites of these sources, though in practice learned RMs are most common due to their scalability. Existing RPDO instantiations are typically one-shot: independently sampling a set of candidates, scoring each once, and selecting the best, analogous to random restarts in traditional optimization. This conservatism is often driven by concerns about reward hacking, favoring shallow selection over deeper optimization. However, shallow sampling does not fully address this concern and faces a dilemma: staying narrow leaves higher-reward responses unrealized, while scaling to wide sampling may exacerbate reward hacking, making downstream selection methods such as best-of-N and self-consistency unreliable (Ichihara et al., 2025). This motivates going beyond independent candidate sampling by searching more deeply and structurally over the response space under a scalar reward objective.

Recent deeper RPDO methods have shown promising gains, but existing approaches are often specialized in both the reward signals and search strategies they employ. Mind Evolution (Lee et al., 2025) emphasizes population-based optimization via genetic algorithms and relies on task-specific programmatic verifiers to score candidates, limiting applicability to settings with reliable automatic evaluators. LLMRefine (Xu et al., 2024) targets open-ended tasks by iteratively refining responses with simulated annealing guided by a natural-language feedback model. The objective is then defined by scalarizing this feedback into a single score. This creates a two-stage pipeline of feedback generation followed by scalarization that may introduce noise and is difficult to calibrate across tasks. Consequently, existing methods are constrained either by requiring verifiable task-specific evaluators or by relying on bespoke feedback modeling and scoring pipelines. Moreover, they typically prioritize either global optimization or local refinement rather than supporting both within a unified procedure. These limitations motivate a memetic RPDO approach that couples global optimization and local refinement under a shared scalar reward signal.

Building on these insights, we present MEMETRON, a memetic RPDO optimizer that formulates response optimization as discrete black-box optimization under scalar reward supervision. MEMETRON alternates between GENETRON for population-based optimization and ANNETRON for annealing-based local refinement within a unified procedure, using frozen LLMs as search operators. The framework requires only scalar reward access, making it compatible with any reward source. In this work, we instantiate MEMETRON with learned RMs, which removes dependence on task-specific programmatic verifiers, costly human evaluation, or task-specific feedback modeling and scoring design, enabling scalable evaluation across diverse tasks (see Figure 1).

Because the search space of RPDO consists of completed natural-language responses, MEMETRON uses off-the-shelf frozen LLMs as variation operators, implementing crossover and mutation for GENETRON and refinement and perturbation for ANNETRON, all directly in response space. These operators are prompted to align with the RM objective without having access to raw reward scores, so that all operator actions are guided by semantic understanding rather than direct score feedback. Components that do not require language understanding, such as population management, annealing schedules, and bookkeeping, are handled by lightweight non-LLM modules driven by scalar reward signals.

We evaluate MEMETRON on instruction-following and mathematical reasoning tasks. MEMETRON reliably discovers higher-scoring responses according to the target RMs, and this translates to downstream gains. On mathematical reasoning, it increases pass@k correctness coverage and improves selection reliability of best-of-N and self-consistency. On instruction following, MEMETRON improves LLM judge preference. On verifiable tasks, we also show that MEMETRON can incorporate ground-truth correctness signals via correctness-conditioned reward shaping, favoring candidates that are both correct and highly scored by the RM. In doing so, comparing runs with and without shaping exposes extreme cases of RM-correctness misalignment, and the resulting response pairs can be served as informative training signal for reward model fine-tuning, rejection sampling SFT warmups for RL-based training pipelines such as PPO and GRPO, and direct preference learning such as DPO.

Our main contributions are:

1. **Problem formulation:** We formulate RPDO as a discrete black-box optimization problem over completed responses,
$$\max_{y \in \mathcal{Y}} r(x, y),$$
unifying existing methods such as best-of-N, SFT with rejection sampling, and DPO with rejection sampling as shallow, one-shot instantiations of this objective. The reward $r(x, y)$ is agnostic to its source and may be instantiated as a learned reward model, programmatic verifier, human feedback, or composite thereof. This framing exposes a gap: existing methods under-exploit the reward signal by treating candidates independently, motivating structured search as a principled alternative.
2. **Memetic optimization framework for RPDO:** We introduce MEMETRON, a memetic response optimizer that alternates between GENETRON for population-based optimization and ANNETRON for annealing-based local refinement under a black-box scalar reward signal. MEMETRON uses frozen LLMs as search operators prompted to align with the reward objective without access to raw reward scores, encouraging operators to improve response quality semantically rather than exploit score artifacts.
3. **MEMETRON outperforms shallow sampling across mathematical reasoning and instruction following tasks:** Across instruction-following and mathematical reasoning tasks, MEMETRON reliably optimizes the target RMs, discovering responses with higher RM scores. Downstream, MEMETRON increases pass@k correctness coverage and the reliability of best-of-N and self-consistency on mathematical reasoning, and improves LLM-judge preference on instruction following.
4. **Reward misalignment diagnostics via correctness shaping:** On verifiable tasks where ground-truth correctness signals are available, MEMETRON can incorporate them via correctness-conditioned reward shaping, favoring candidates that are both correct and highly scored by the RM. This strengthens RM-based selection toward the Pass@$|H|$ upper bound and, by comparing runs with and without shaping, exposes RM-correctness misalignment, yielding contrastive response pairs that can support reward model fine-tuning, rejection sampling SFT warmups for RL-based training pipelines such as PPO and GRPO, and direct preference learning such as DPO.

2 Problem Formulation

We formulate RPDO as a discrete black-box optimization problem over completed LLM responses, without updating the model parameters, with the objective of identifying a high-quality response under a given scalar

evaluator. Formally, given a prompt $x \in \mathcal{X}$, a set of frozen LLMs $\{\pi_m(y \mid x)\}_{m=1}^{M}$ (with $M \geq 1$), and a black-box scalar evaluator $r : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ which assigns a scalar reward to each response $y \in \mathcal{Y}$, the goal is to find

$$y^{\dagger} \;=\; \arg\max_{y \in \mathcal{Y}} r(x, y),$$

where the response space $\mathcal{Y}$ consists of all sequences producible by the frozen models $\{\pi_m(y \mid x)\}_{m=1}^{M}$ under any decoding strategy. The reward function r is agnostic to its source and may be instantiated as a model-derived score such as the sequence log-likelihood under a frozen model, a learned reward model, a programmatic verifier, a simulator, human feedback, rubric-based scoring, or a composite of these sources.

This setup presents several challenges:

- $\mathcal{Y}$ is discrete and finite but astronomically large, growing exponentially with the vocabulary size $|V|$ and the maximum length $\ell_{\max}$, on the order of $|V|^{\ell_{\max}}$,
- $r(x, y)$ is a black-box scalar reward signal with no gradient information, and
- the objective landscape is highly non-smooth and multimodal, where small changes in y can cause abrupt jumps in r and distinct high-reward regions may be disconnected.

Because the search space is discrete, non-differentiable, and highly multimodal, gradient-based methods and exact combinatorial search are ill-suited. We instead turn to memetic algorithms (MA), which combine population-based exploration via genetic algorithms (GA) with local refinement via simulated annealing (SA) and require only black-box queries to $r(x, y)$. Our goal is not to exhaustively search $\mathcal{Y}$ or guarantee global optimality, which is infeasible under realistic inference budgets. Instead, we aim to efficiently discover responses with higher reward than those obtained by standard decoding or shallow reranking.

Reward Models While RPDO is compatible with any black-box scalar evaluator, we primarily focus on learned RMs in this work, as they capture task-relevant aspects of response quality that are otherwise difficult to specify without hand-crafted reward functions. Unlike programmatic verifiers, which are limited to tasks with easily verifiable answers, learned RMs are broadly applicable across tasks. Once trained on human preference annotations, they efficiently approximate human judgment at inference scale without per-instance human evaluation. Furthermore, they do not require task-specific evaluation infrastructure such as manual scalarization of natural language feedback. Learned RMs range from single-attribute scorers targeting a specific dimension to general preference models capturing multiple aspects of quality, and MEMETRON is compatible with both: for specific RMs, operators are prompted to align with the targeted dimension; for general RMs, the same RM can be reused across different objectives by adjusting the operator prompts alone, without retraining or replacing the RM.

Despite many advantages, RMs are inherently imperfect and may be partially misaligned with downstream goals, exhibiting failures in edge cases and susceptibility to reward hacking. That said, learning robust reward and preference models is a well-studied and active problem. Prior work has examined what makes RMs effective learning signals (Razin et al., 2025), how over-optimization and reward hacking can arise (Gao et al., 2023), and how such failures may be mitigated through constrained objectives (Moskovitz et al., 2023), demonstration-guided methods (Rita et al., 2024), or data-centric approaches (Nguyen et al., 2025; Northcutt et al., 2021). This work does not assume perfect reward alignment; rather, we assume access to a reasonably informative and robust RM, treating robust RM design as an orthogonal concern addressed by the literature above.

3 MEMETRON: Memetic Optimization for Reward-Guided Post-Decoding Response Optimization

We propose MEMETRON, a memetic RPDO optimizer that uses frozen LLMs as search operators prompted to align with the reward objective without access to raw reward scores, enabling them to propose and execute targeted edits and syntheses guided by semantic understanding of the objective. MEMETRON alternates

between GENETRON for population-based optimization and ANNETRON for annealing-based local refinement, with lightweight non-LLM modules handling population management and annealing schedules via scalar reward signals.

At each generation g, MEMETRON maintains a population $G^{(g)} = \{y_i^{(g)}\}_{i=1}^N$ and a history buffer $\mathcal{H}$ containing all previously evaluated candidates. The initial population is obtained by sampling from the base model,

$$y_i^{(0)} \sim \pi(y \mid x).$$

Each generation proceeds in three stages: global optimization, local refinement, and population selection. First, GENETRON applies ***LLM-as-Crossover-Operator*** and ***LLM-as-Mutation-Operator*** to the current population to generate offspring candidates. Second, each offspring is refined by ANNETRON, which performs reward-guided local edits using ***LLM-as-Refinement-Operator*** and ***LLM-as-Perturbation-Operator***. In both stages, the LLM operators are prompted to align with the reward objective without access to raw reward scores; full details are given in Sections 3.1 and 3.2. Third, all newly generated candidates are added to the history buffer, and the next population is selected as the top-N responses by reward:

$$G^{(g+1)} = \arg \max_{\substack{S \subseteq \mathcal{H} \\ |S|=N}} \sum_{y \in S} r(x, y).$$

This elitist selection strategy preserves the best solutions discovered so far while allowing exploration through stochastic generation operators. The search process terminates after a fixed number of generations or when improvement plateaus. The final output is selected as

$$\hat{y} = \arg \max_{y \in \mathcal{H}} r(x, y).$$

Algorithm 1 MEMETRON: Memetic Optimization for RPDO

$g \leftarrow 0$
Initialize population $G^{(0)} = \{y_1^{(0)}, \ldots, y_N^{(0)}\}$ by sampling from model $\pi(y \mid x)$
Evaluate rewards and initialize history buffer: $\mathcal{H} \leftarrow G^{(0)}$
while termination criterion not met **do**
  Apply GENETRON's genetic operators to $G^{(g)}$ to produce offspring $O^{(g)} = \{y_1^*, \ldots, y_N^*\}$
  for all $y^* \in O^{(g)}$ **do**
    Refine y^* using ANNETRON with respect to reward r
  end for
  Update history: $\mathcal{H} \leftarrow \mathcal{H} \cup O^{(g)}$
  Form next population:

$$G^{(g+1)} = \arg \max_{\substack{S \subseteq \mathcal{H} \\ |S|=N}} \sum_{y \in S} r(x, y)$$

  $g \leftarrow g + 1$
end while
Output: $\mathcal{H}$ and $\hat{y} = \arg\max_{y \in \mathcal{H}} r(x, y)$

3.1 GENETRON: Genetic Optimization for RPDO

We introduce GENETRON, a population-based evolutionary search procedure for RPDO inspired by genetic algorithms. The process begins by initializing a population $G^{(0)} = \{y_1^{(0)}, \ldots, y_N^{(0)}\} \subset \mathcal{Y}$ of N candidate responses, sampled from one or multiple models $\{\pi_m(y \mid x)\}_{m=1}^M$, with M denoting the total number of models and x being the input prompt. Each response $y_i^{(0)} \in G^{(0)}$ is evaluated using the scalar evaluator $r(x, y)$, yielding scores $R^{(0)} = \{r(x, y_1^{(0)}), \ldots, r(x, y_N^{(0)})\}$.

At each subsequent generation g, a parent set $P^{(g)} \subseteq G^{(g)}$ is selected via binary tournament selection: for each parent, two candidates are sampled randomly from the current population, and the one with the higher reward $r(x, y)$ is retained. To generate offspring, we apply ***crossover*** via ***LLM-as-Crossover-Operator***. For each offspring, a pair of parents $y_i, y_j \in P^{(g)}$ is sampled uniformly at random, and and a crossover prompt

$x_{\text{cross}}(x, y_i, y_j)$ is constructed to condition on the original input prompt x and the two parent responses. Concretely, x_{cross} instructs the LLM to (i) carefully analyze the user query and both parent responses with respect to task requirements and reward-relevant criteria, and (ii) produce a structured synthesis plan describing how to combine the strongest complementary elements from y_i and y_j into an improved solution. x_{cross} is designed to emphasize properties targeted by $r(x, y)$, thereby aligning crossover generation with the optimization objective. A crossover draft is then sampled as

$$y^{\text{cross}} \sim \pi(y \mid x_{\text{cross}}(x, y_i, y_j)) .$$

Next, ***mutation*** is performed via the ***LLM-as-Mutator-Operator***, which refines the crossover draft while remaining grounded in the original prompt and the parent solutions. Specifically, a mutation prompt $x_{\text{mut}}(x, y_i, y_j, y^{\text{cross}})$ is constructed from x, the two parents y_i, y_j, and the crossover draft y^{cross}. The mutation prompt instructs the LLM to treat the crossover draft as a primary guide and to produce a single final response that best satisfies the task requirements in x. Analogously, x_{mut} is crafted to encourage edits that improve reward under $r(x, y)$, aligning mutation with the same objective. From each mutation prompt, we draw n mutated candidates,

$$\{y_1^{\text{mut}}, \ldots, y_n^{\text{mut}}\} \sim \pi(y \mid x_{\text{mut}}(x, y_i, y_j, y^{\text{cross}})) ,$$

which implements multi-try stochastic mutation within the neighborhood induced by x_{mut}, reducing sensitivity to any single sample and increasing the chance of escaping low-reward local edits. We select the highest-reward mutation:

$$y^* = \underset{k \in \{1,\ldots,n\}}{\arg\max}\ r\big(x, y_k^{\text{mut}}\big) .$$

Prompt templates for both ***LLM-as-Crossover-Operator*** and ***LLM-as-Mutator-Operator*** operators are provided in Appendix D for both instruction-following and mathematical-reasoning tasks.

The resulting set of all generated offsprings y^* of generation g is denoted as $O^{(g)}$. Throughout the process, we maintain a cumulative search history

$$\mathcal{H} = G^{(0)} \cup \bigcup_{g=1}^{L} O^{(g)},$$

where L stands for the last generation index. The next population $G^{(g+1)}$ is formed with ***elitism***, by selecting the top N responses from the full history $\mathcal{H}^{(g)}$, based on reward:

$$G^{(g+1)} = \underset{\substack{S \subseteq \mathcal{H}^{(g)} \\ |S| = N}}{\arg\max} \sum_{y \in S} r(x, y).$$

This step allows high-reward individuals from any previous generation to persist if they outperform newly generated candidates.

The evolutionary process continues until a convergence criterion is met, such as hitting a fixed computational budget (e.g., total number of model calls), reaching a predefined reward threshold, or a fixed number of generations L is reached. In practice, convergence is determined by monitoring the improvement in maximum reward across generations. Specifically, if the best reward in the current population $G^{(g)}$ has not improved by more than a small threshold δ over the past L_{patience} generations, the search is considered to have converged. Formally, we stop if

$$\max_{y \in G^{(g)}} r(x, y) - \max_{y \in G^{(g - L_{\text{patience}})}} r(x, y) < \delta.$$

This patience-based criterion prevents unnecessary computation once the search plateaus and further improvements are unlikely. The highest quality response for a given prompt can then be extracted from the history by computing:

$$\hat{y} = \arg\max_{y \in \mathcal{H}} r(x, y)$$

Algorithm 2 GENETRON: Genetic Optimization for RPDO

Initialize $g \leftarrow 0$
Initialize population $G^{(0)} = \{y_1^{(0)}, \ldots, y_N^{(0)}\}$ by sampling from model $\pi(y \mid x)$
Evaluate rewards: $R^{(0)} = \{r(x, y_1^{(0)}), \ldots, r(x, y_N^{(0)})\}$
Initialize history buffer: $\mathcal{H}^{(0)} \leftarrow G^{(0)}$
while termination criterion not met **do**
  Selection: Select parents $P^{(g)} \subseteq G^{(g)}$ via binary tournament selection
  Initialize offspring set: $O^{(g)} \leftarrow \emptyset$
  for $k = 1$ to N **do**
    Crossover: Sample parent pair $y_i, y_j \in P^{(g)}$
    Construct prompt $x_{\text{cross}} \leftarrow x_{\text{cross}}(x, y_i, y_j)$
    Sample crossover draft $y^{\text{cross}} \sim \pi(y \mid x_{\text{cross}})$
    Mutation: Construct prompt $x_{\text{mut}} \leftarrow x_{\text{mut}}(x, y_i, y_j, y^{\text{cross}})$
    Sample n candidates $\{y_1^{\text{mut}}, \ldots, y_n^{\text{mut}}\} \sim \pi(y \mid x_{\text{mut}})$
    Select best candidate $y^* = \underset{k \in \{1,\ldots,n\}}{\arg\max}\ r(x, y_k^{\text{mut}})$
    Add y^* to $O^{(g)}$
  end for
  Update history: $\mathcal{H}^{(g+1)} \leftarrow \mathcal{H}^{(g)} \cup O^{(g)}$
  Elitism: Form next population

$$G^{(g+1)} = \underset{\substack{S \subseteq \mathcal{H}^{(g+1)} \\ |S| = N}}{\arg\max} \sum_{y \in S} r(x, y)$$

  $g \leftarrow g + 1$
end while
Output: $\mathcal{H}$ and best response $\hat{y} = \arg\max_{y \in \mathcal{H}} r(x, y)$

Algorithm 3 ANNETRON: Simulated Annealing for RPDO

Initialize $t \leftarrow 0$
Sample initial response $y_0 \sim \pi(y \mid x)$
Evaluate reward $r_0 = r(x, y_0)$
Set temperature $T_0 > 0$, decay factor $\alpha \in (0, 1)$
Initialize history buffer $\mathcal{H} \leftarrow \{y_0\}$
while termination criterion not met **do**
  Refinement: Construct prompt $x_{\text{refine}} \leftarrow x_{\text{refine}}(x, y_t)$
  Sample refinement draft $y_t^{\text{refine}} \sim \pi(y \mid x_{\text{refine}})$
  Perturbation : Construct prompt $x_{\text{pert}} \leftarrow x_{\text{pert}}(x, y_t, y_t^{\text{refine}})$
  Sample n candidates $\{y_{t,1}^{\text{pert}}, \ldots, y_{t,n}^{\text{pert}}\} \sim \pi(y \mid x_{\text{pert}})$
  Select best candidate $y_t^* = \underset{k \in \{1,\ldots,n\}}{\arg\max}\ r(x, y_{t,k}^{\text{pert}})$
  Compute reward difference $\Delta r = r(x, y_t^*) - r(x, y_t)$
  if $\Delta r \geq 0$ **then**
    Accept: $y_{t+1} \leftarrow y_t^*$
  else
    Accept with probability $p = \exp(\Delta r / T_t)$
    if `random()` $< p$ **then**
      $y_{t+1} \leftarrow y_t^*$
    else
      $y_{t+1} \leftarrow y_t$
    end if
  end if
  Update temperature: $T_{t+1} \leftarrow \alpha T_t$
  if $y_{t+1} \neq y_t$ **then**
    Update history: $\mathcal{H} \leftarrow \mathcal{H} \cup \{y_{t+1}\}$
  end if
  $t \leftarrow t + 1$
end while
Output: $\mathcal{H}$ and best response $\hat{y} = \arg\max_{y \in \mathcal{H}} r(x, y)$

Remark 1. *GENETRON implements crossover and mutation via the* ***LLM-as-Crossover-Operator*** *and* ***LLM-as-Mutation-Operator*** *rather than hand-crafted combinatorial operators, enabling semantically coherent offspring generation directly in natural-language response space. These operators are coupled in a two-stage pipeline: the crossover operator analyzes the current parent responses and produces a synthesis plan and draft, which the mutation operator then refines. Both operators are prompted to align with the r objective without exposing scalar reward scores to the LLM, so that candidate generation is guided by semantic understanding rather than direct score feedback. Per offspring, a small number n of mutation candidates are sampled to simulate stochastic mutation, and the highest-reward one is retained. This differs from naive best-of-n in that candidates are not independently sampled from the base LLM but are conditioned on the crossover draft and parent responses, making them semantically informed proposals rather than random draws. Unlike task-specific evolutionary approaches that rely on programmatic verifiers, all selection, elitism, and convergence decisions of GENETRON are driven exclusively by scalar values $r(x, y)$, keeping it reward-agnostic and broadly applicable across tasks and r instantiations.*

3.2 ANNETRON: Simulated Annealing for RPDO

We introduce ANNETRON, a local refinement procedure for RPDO inspired by simulated annealing. The optimization process begins by initializing a candidate response $y_0 \in \mathcal{Y}$, sampled from a conditional LLM $\pi(y \mid x)$, where x is the input prompt. The corresponding reward is computed as $r_0 = r(x, y_0)$. A temperature parameter $T_0 > 0$ is also initialized, and the search proceeds over discrete time steps indexed by t.

At each iteration t, ***LLM-as-Refinement-Operator*** is first applied to the current candidate y_t to derive a targeted improvement blueprint. In particular, a refinement prompt $x_{\text{refine}}(x, y_t)$ is constructed to instruct the LLM to (i) carefully analyze the user query x and the current response y_t with respect to task requirements and reward-relevant criteria, and (ii) produce a structured refinement plan that specifies concrete edits and improvements to improve $r(x, y)$. The resulting refinement artifact is sampled as

$$y_t^{\text{refine}} \sim \pi(y \mid x_{\text{refine}}(x, y_t)) .$$

Next, a perturbed candidate is produced via an ***LLM-as-Perturbation-Operator***, which is analogous to the mutation operator used in population-based methods but operates on a *single* current response. A perturbation prompt $x_{\text{pert}}(x, y_t, y_t^{\text{refine}})$ is constructed from the original prompt x, the current candidate y_t, and the refinement draft y_t^{refine}. The prompt instructs the LLM to treat y_t^{refine} as the primary guide and to produce a single candidate response that best satisfies the task requirements in x, while making targeted improvements relative to y_t. From the perturbation prompt, n candidates are sampled:

$$\{y_{t,1}^{\text{pert}}, \ldots, y_{t,n}^{\text{pert}}\} \sim \pi\big(y \mid x_{\text{pert}}(x, y_t, y_t^{\text{refine}})\big)\,,$$

which provides an implicit local random-restart effect within the perturbation neighborhood induced by x_{pert}, improving robustness and increasing the chance of escaping low-reward local edits. We select the highest-reward candidate:

$$y_t^* = \underset{k\in\{1,\ldots,n\}}{\arg\max}\; r\Big(x, y_{t,k}^{\text{pert}}\Big)\,.$$

Prompt templates for both the ***LLM-as-Refinement-Operator*** and ***LLM-as-Perturbation-Operator*** operators are provided in Appendix D for both instruction-following and mathematical-reasoning tasks.

We then compute the reward difference as $\Delta r = r(x, y^*) - r(x, y_t)$, which determines whether the new candidate is accepted. If y^* improves upon y_t (i.e., $\Delta r \geq 0$), it is always accepted. If the reward decreases ($\Delta r < 0$), the new candidate may still be accepted with a probability given by the *Metropolis criterion*:

$$P(\text{accept}) = \begin{cases} 1, & \text{if } \Delta r \geq 0 \\ \exp\left(\dfrac{\Delta r}{T_t}\right), & \text{if } \Delta r < 0, \end{cases}$$

where temperature T_t is updated at each step using a geometric decay schedule:

$$T_{t+1} = \alpha T_t, \quad \text{with } \alpha \in (0, 1).$$

This probabilistic acceptance criterion allows non-greedy exploration, enabling the search to escape local optima and increasing the likelihood of discovering globally optimal responses. At high temperatures, ANNETRON explores more widely, frequently accepting worse responses to traverse the response landscape. As the temperature decreases, the algorithm becomes increasingly greedy, primarily accepting responses that yield reward improvements.

Throughout the optimization process, we maintain a history set $\mathcal{H}$ of all accepted responses, beginning with the initial candidate and including each accepted refinement:

$$\mathcal{H} = \{y_0\} \cup \{y_1^*, y_2^*, \ldots, y_L^*\},$$

where each y_t^* denotes a response accepted at iteration t, and L is the last iteration index.

The process continues for a fixed number of steps or until a termination condition is met (e.g., reward plateau or convergence). The best response observed across all accepted candidates is returned as the final output:

$$\hat{y} = \underset{y\in\mathcal{H}}{\arg\max}\; r(x, y).$$

Remark 2. *ANNETRON implements local refinement via the* ***LLM-as-Refinement-Operator*** *and* ***LLM-as-Perturbation-Operator****, enabling semantically coherent local search directly in natural-language response space. These operators are coupled in a two-stage pipeline: the refinement operator analyzes the current candidate and produces a targeted improvement plan, which the perturbation operator then uses to generate a perturbed candidate. Both operators are prompted to align with the r objective without exposing scalar reward values $r(x, y)$ to the LLM, so that candidate generation is guided by semantic understanding rather than direct score feedback. Per step, a small number n of perturbation candidates are sampled to simulate stochastic perturbation, and the highest-reward candidate under r is used as the proposal for the Metropolis acceptance criterion, which allows non-greedy exploration and escape from local optima. Unlike approaches that rely on natural language feedback models or task-specific evaluation pipelines, ANNETRON operates under any scalar reward signal, keeping it reward-agnostic and broadly applicable across tasks and r instantiations.*

4 Experiments

4.1 Mathematical reasoning

Dataset. We curated 42 non-figure problems from the 2025 AMC 12 A and B tests, administered in November 2025; problem statements and answer keys were obtained from Art of Problem Solving (Art of Problem Solving, 2026). AMC 12 is a U.S. national mathematics competition for high-school students presented in a multiple-choice format.

Generator details. We generate responses with Qwen3-8B (Yang et al., 2025) in non-thinking mode using temperature = 1.5, min-p = 0.1, top-k = 50, and a maximum generation length of 16,384 tokens. For all experiments, we provide the model with problem statements only, omitting the multiple-choice options to encourage free-form reasoning rather than elimination-based strategies. All generations are run on a single NVIDIA A100 (80 GB). Because the 2025 AMC 12 postdates Qwen3-8B's April 2025 release, this benchmark reduces the risk of training-data contamination. All LLM operators of our proposed methods use the same model and generation settings.

Reward model details. MEMETRON and its algorithms are designed to work with any black-box scalar reward signal, making them compatible with general-purpose reward models (RMs) such as the Skywork-Reward-V2 family, which can be reused across tasks by changing only the operator prompts. In this experiment, we infer reward signals r using Skywork-Reward-V2-Llama-3.1-8B-40M (Liu et al., 2025), a higher-capacity variant in the Skywork-Reward-V2 discriminative outcome RM family that performs strongly on RewardBench v2 (Malik et al., 2025). Reward inference is served through an internal Gradio API hosted on a single NVIDIA A100 (80 GB).

Baselines. Base-16 consists of 16 independently sampled responses. To separate the effect of our proposed algorithms from simply increasing the sampling budget, we also report pure-sampling baselines Base-128 and Base-1024.

Proposed RPDO methods. We evaluate all three of our proposed deep RPDO algorithms: ANNETRON, GENETRON, and MEMETRON. All methods are initialized from Base-16. ANNETRON runs for five iterations and GENETRON runs for five generations; in both cases, each step retains exactly 16 candidates. Over five steps, this yields a final candidate buffer H with $|H| = 96$. At each step, the **LLM-as-Perturbation-Operator** in ANNETRON and the **LLM-as-Mutation-Operator** in GENETRON each sample $n = 3$ candidates conditioned on the refinement draft and crossover draft, respectively, and retain the highest-reward candidate. MEMETRON combines both operators, running for five rounds, where each round consists of one GENETRON generation followed by five ANNETRON iterations. All LLM operators are prompted to align with the mathematical reasoning objective without access to numerical reward scores.

To isolate the effect of correctness-signal augmentation, we additionally evaluate MEMETRON with *correctness shaping*, which modifies the reward as $r' = r + c \cdot \mathbb{1}[\text{answer is correct}]$, where r is the base reward score and $c = 20$. All other aspects of the optimization procedure remain identical to MEMETRON. Although the proposed methods support reward-based early stopping, we disable adaptive stopping to ensure controlled and comparable evaluation across methods. Prompt templates for the LLM operators are provided in Appendix D.2.

Evaluation metrics and protocol. We evaluate performance using four metrics: *Pass@*$|H|$ accuracy, *Best RM*, *Best-of-*$|H|$ accuracy, and *self-consistency* accuracy. Here, $|H|$ denotes the size of the history buffer accumulated by an algorithm up to that generation; *Pass@*$|H|$ and *Best-of-*$|H|$ are therefore instantiations of the standard *Pass@k* and *Best-of-N* metrics with $k = N = |H|$. *Pass@*$|H|$ measures whether at least one correct solution appears in the candidate pool and serves as an upper-bound indicator of correctness coverage; it is most useful when automatic answer verification is cheap and reliable. *Best RM* is the highest reward score observed in H, reported as the mean across problems with 95% confidence intervals. *Best-of-*$|H|$ accuracy and *self-consistency* accuracy measure the accuracy of the highest-reward answer and the most

frequent answer in H, respectively, both reflecting a deployment-realistic selection rule. Full definitions of *Pass@k*, *Best-of-N*, and *self-consistency* are provided in Appendix E.

Results. Correctness coverage under automatic verification. The underlying generator is capable of producing correct solutions for nearly all problems, but requires wide sampling to do so: Base-128 reaches 97.62% and Base-1024 achieves 100% Pass@$|H|$ across all 42 problems.

Starting from the same Base-16 initialization (85.71%), the proposed deep RPDO methods guide the generator toward higher-coverage solutions *without access to ground-truth verification during optimization*: ANNETRON reaches 90.48%, GENETRON remains at 85.71%, and MEMETRON increases to 95.24%. To assess the ceiling of RPDO under ideal reward conditions, MEMETRON with correctness shaping simulates a perfect reward model by incorporating ground-truth correctness into the reward signal, reaching 97.62%.

However, Pass@$|H|$ itself requires oracle verification, which is rarely available in practice: programmatic verification is not generally available for open-ended mathematical reasoning, manual verification across hundreds of responses is highly impractical, and RMs are not yet reliable enough to substitute for ground truth.

Proxy-based selection without automatic verification. In the absence of ground-truth verification, practitioners must rely on imperfect proxies: RM-based selection such as Best-of-N, and aggregation heuristics such as self-consistency. However, under pure sampling, simply increasing the response pool is counterproductive: as the pool grows, reward misranking is exacerbated by the RM's imperfect calibration over an increasingly diverse set of responses. Best RM raises monotonically ($25.64 \rightarrow 29.90 \rightarrow 32.92$ for Base-16, Base-128, and Base-1024), yet selection accuracy degrades: Best-of-$|H|$ accuracy drops from 61.90% to 54.76% to 47.62%, indicating that larger response pools increasingly surface high-reward but incorrect responses; and self-consistency accuracy remains at 69.05% for both Base-128 and Base-1024, suggesting that additional responses yield diminishing returns for majority-based aggregation.

Our proposed RPDO methods mitigate this failure mode by optimizing the response distribution against the RM, rather than merely expanding pool size. At $|H| = 96$, ANNETRON and GENETRON achieve Best RM scores of 32.21 and 34.01, with Best-of-$|H|$ and self-consistency accuracies reaching up to 80.95% and 85.71% respectively. MEMETRON yields the strongest overall performance, reaching a Best RM of 37.48 with Best-of-$|H|$ and self-consistency accuracies of 85.71% and 88.10%, indicating that deeper, multi-step RPDO produces response sets that are both higher-scoring and more reliably selected under proxy supervision. Notably, all proposed methods attain non-overlapping 95% confidence intervals in Best RM relative to Base-16: ANNETRON from iteration four onward, GENETRON from generation two onward, and MEMETRON from the first generation onward, providing evidence that the observed reward improvements are statistically reliable.

Reward recognition bottleneck. Despite these substantial gains, a gap between Pass@$|H|$ and Best-of-$|H|$ accuracy persists: for MEMETRON at five generations, correct solutions exist for 40 of 42 problems (95.24%), yet the RM fails to rank them highest for 4 of those, yielding a Best-of-$|H|$ of 36/42 (85.71%). Best RM of MEMETRON continues to improve beyond Base-1024 (37.48 vs. 32.92), suggesting that RPDO finds extreme high-reward but incorrect responses beyond the reach of pure sampling.

To address this, MEMETRON with correctness shaping augments the RM signal with a 20-point bonus for correct responses, steering the model toward responses that are both high-reward and correct. While Best RM reaches 55.29, this reflects the 20-point bonus; accounting for it, the underlying RM scores are comparable to unaugmented MEMETRON, yet the highest-reward responses are now correct for problems where the generator can produce correct answers, and therefore selected by Best-of-$|H|$. As a result, accuracies reach 97.62% and 95.24% for Best-of-$|H|$ and self-consistency after five generations, closing the gap with Pass@$|H|$ (97.62%).

The contrastive pairs produced by MEMETRON with and without correctness shaping provide a concrete diagnostic and training resource, enabling reward model fine-tuning, rejection sampling SFT warmups for RL-based training pipelines such as PPO and GRPO, and direct preference learning such as DPO, which we analyze further in Appendix G.3.

Table 1: Performance of sampling baselines and proposed RPDO methods on 42 non-figure AMC 12 2025 problems, using Qwen3-8B (non-thinking) as the generator and Skywork-Reward-V2-Llama-3.1-8B-40M as the reward model. Base-16 is the shared initialization pool for all RPDO methods; Base-128 and Base-1024 are pure sampling baselines to illustrate their limitations. Each RPDO generation appends 16 new candidates to the cumulative pool, so all methods at five generations reach $|H| = 96$. MEMETRON with correctness shaping is an oracle variant that augments the reward signal with ground-truth correctness. Metric definitions are in Appendix E; experimental details in Section 4.1.

System	**Pass@$\|H\|$**	**Best RM ($\mathbf{CI_{95}}$)**	**Best-of-$\|H\|$**	**Self-consistency**	$\|H\|$
Base-16	36/42 (85.71%)	25.64 [22.94, 28.34]	26/42 (61.90%)	30/42 (71.43%)	16
Base-128	41/42 (97.62%)	29.90 [27.69, 32.11]	23/42 (54.76%)	29/42 (69.05%)	128
Base-1024	42/42 (100.00%)	32.92 [30.81, 35.04]	20/42 (47.62%)	29/42 (69.05%)	1024
ANNETRON (Base-16 + 1 iteration)	38/42 (90.48%)	28.25 [25.65, 30.86]	33/42 (78.57%)	31/42 (73.81%)	32
ANNETRON (Base-16 + 2 iterations)	38/42 (90.48%)	30.29 [27.80, 32.77]	33/42 (78.57%)	33/42 (78.57%)	48
ANNETRON (Base-16 + 3 iterations)	38/42 (90.48%)	30.73 [28.24, 33.21]	33/42 (78.57%)	34/42 (80.95%)	64
ANNETRON (Base-16 + 4 iterations)	38/42 (90.48%)	31.60 [29.17, 34.03]	34/42 (80.95%)	34/42 (80.95%)	80
ANNETRON (Base-16 + 5 iterations)	38/42 (90.48%)	32.21 [29.86, 34.57]	34/42 (80.95%)	34/42 (80.95%)	96
GENETRON (Base-16 + 1 generation)	36/42 (85.71%)	29.47 [27.18, 31.76]	33/42 (78.57%)	33/42 (78.57%)	32
GENETRON (Base-16 + 2 generations)	36/42 (85.71%)	31.23 [28.92, 33.54]	34/42 (80.95%)	35/42 (83.33%)	48
GENETRON (Base-16 + 3 generations)	36/42 (85.71%)	32.25 [29.91, 34.59]	35/42 (83.33%)	36/42 (85.71%)	64
GENETRON (Base-16 + 4 generations)	36/42 (85.71%)	33.40 [30.98, 35.83]	35/42 (83.33%)	36/42 (85.71%)	80
GENETRON (Base-16 + 5 generations)	36/42 (85.71%)	34.01 [31.62, 36.40]	35/42 (83.33%)	36/42 (85.71%)	96
MEMETRON (Base-16 + 1 generation)	39/42 (92.86%)	33.39 [31.01, 35.78]	33/42 (78.57%)	36/42 (85.71%)	32
MEMETRON (Base-16 + 2 generations)	39/42 (92.86%)	34.62 [32.25, 36.99]	35/42 (83.33%)	36/42 (85.71%)	48
MEMETRON (Base-16 + 3 generations)	40/42 (95.24%)	35.74 [33.39, 38.08]	35/42 (83.33%)	36/42 (85.71%)	64
MEMETRON (Base-16 + 4 generations)	40/42 (95.24%)	36.81 [34.60, 39.03]	35/42 (83.33%)	36/42 (85.71%)	80
MEMETRON (Base-16 + 5 generations)	**40/42 (95.24%)**	**37.48 [35.32, 39.64]**	**36/42 (85.71%)**	**37/42 (88.10%)**	96
MEMETRON + Shaping (Base-16 + 1 generation)	39/42 (92.86%)	49.74 [46.30, 53.18]	39/42 (92.86%)	36/42 (85.71%)	32
MEMETRON + Shaping (Base-16 + 2 generations)	39/42 (92.86%)	51.74 [48.59, 54.90]	39/42 (92.86%)	38/42 (90.48%)	48
MEMETRON + Shaping (Base-16 + 3 generations)	41/42 (97.62%)	53.46 [50.58, 56.34]	41/42 (97.62%)	39/42 (92.86%)	64
MEMETRON + Shaping (Base-16 + 4 generations)	41/42 (97.62%)	54.50 [51.61, 57.39]	41/42 (97.62%)	39/42 (92.86%)	80
MEMETRON + Shaping (Base-16 + 5 generations)	**41/42 (97.62%)**	**55.29 [52.43, 58.15]**	**41/42 (97.62%)**	**40/42 (95.24%)**	96

Compute-quality tradeoff and diminishing returns. The improvements of our algorithms come at the cost of increased inference time. Base-16, Base-128, and Base-1024 require 34.19, 93.13, and 601.13 seconds per question, respectively. At five generations, ANNETRON requires 434.31 seconds per question, GENETRON 491.10 seconds, and MEMETRON 2318.45 seconds; MEMETRON with correctness shaping incurs similar cost at 2381.48 seconds per question. The higher cost of MEMETRON reflects its nested schedule: each round runs one GENETRON generation followed by five ANNETRON iterations. All times are problem-dependent, as harder problems tend to induce longer generations. Because this benchmark is relatively challenging, the reported runtimes likely skew toward the upper end of what we observe in easier settings. Measurements include response generation, reward-model scoring, network latency from the Gradio API, and optimization overhead, and reflect a sequential Python implementation on fixed hardware without parallelization or runtime optimizations. Complementary trajectory analysis in Appendix F.1 shows diminishing returns across questions and suggests that reward-based early stopping can substantially reduce average cost in deployment.

4.2 Instruction Following

Dataset. To evaluate the effectiveness of our proposed methods on instruction-following tasks, we use the TinyAlpacaEval dataset (Polo et al., 2024), which comprises 100 examples selected from the full 805-example AlpacaEval 2.0 benchmark (Li et al., 2023) via Item Response Theory (IRT) to maximize informativeness while preserving the core evaluation properties of the original benchmark.

Generator details. For the instruction-following experiment, we use Llama-3.2-3B-Instruct. Responses are generated with sampling hyperparameters: temperature $= 1.5$, min-$p = 0.1$, top-$k = 50$, and a maximum generation length of 4,096 tokens. All experiments are conducted on a single NVIDIA A100 (80 GB). All LLM operators of the proposed method in this experiment also use the same model and generation settings.

Reward model details. As with the mathematical reasoning experiments, MEMETRON utilizes a general-purpose RM from the Skywork-Reward-V2 family. In this experiment, r signals are inferred using Skywork-Reward-V2-Llama-3.2-3B (Liu et al., 2025). We use the 3B variant to keep the reward model at a similar scale to the base generator. Inference is served via an internal Gradio API hosted on a single NVIDIA A100 GPU (80 GB).

MEMETRON. We run MEMETRON initialized from Base-16 for five rounds, where each round consists of one generation of GENETRON followed by five iterations of ANNETRON, with LLM operators prompted to align with the instruction-following objective without access to numerical reward scores. At each step, both the LLM-as-Perturbation-Operator in ANNETRON and the LLM-as-Mutation-Operator in GENETRON sample $n = 3$ candidates, retaining the highest-reward one. Each generation retains 16 candidates, yielding a history buffer H with $|H| = 96$ responses. Instruction following prompt templates for the operators are provided in Appendix D.3.

Evaluation metrics and protocol. We evaluate performance using four metrics: *Best RM*, marginal gain Δ_g, *Preference* win rate, and *Margin*. $|H|$ denotes the size of the cumulative candidate pool available up to generation g (including the Base-16 initialization). *Best RM* is the highest reward score observed in H, reported as the mean across prompts with 95% confidence intervals. The marginal gain is defined as $\Delta_g = \text{best}(g) - \text{best}(g-1)$, where $\text{best}(g)$ denotes the best RM score in the cumulative pool at generation g; we report the mean Δ_g across prompts with 95% bootstrap confidence intervals. To complement RM-based evaluation, we additionally compare the *Best RM*-selected response against the Base-16 baseline using GPT-5-nano with two presentation orders (A/B and B/A; 5 judgments each, 10 total per prompt), reporting tie-aware *Preference* and *Margin* averaged across prompts with 95% hierarchical bootstrap confidence intervals (Appendix E).

Results. Table 2 reports MEMETRON performance across generations on TinyAlpacaEval. Best RM increases monotonically from 16.60 at Base-16 to 19.62 after five generations, with non-overlapping confidence intervals relative to Base-16 from generation 3 onward, and mean marginal gains diminishing from 1.34 at

Table 2: Performance of MEMETRON on 100 TinyAlpacaEval prompts, using Llama-3.2-3B-Instruct as the generator and Skywork-Reward-V2-Llama-3.2-3B as the reward model. Base-16 is the shared initialization pool; each additional generation appends 16 new candidates to the cumulative pool, so MEMETRON at five generations reaches $|H| = 96$. Metric definitions are in Appendix E; experimental details in Section 4.2.

System	Best RM (CI_{95})	Mean Δ (CI_{95})	Preference (CI_{95})	Margin (CI_{95})	$\lvert H\rvert$
Base-16	16.60 [15.37, 17.82]	—	—	—	16
MEMETRON (Base-16 + 1 generation)	17.93 [16.74, 19.13]	1.34 [0.98, 1.72]	58.65% [53.35%, 64.25%]	+17.30 pp [+06.70 pp, +28.50 pp]	32
MEMETRON (Base-16 + 2 generations)	18.56 [17.36, 19.76]	0.62 [0.47, 0.78]	63.80% [57.70%, 69.85%]	+27.60 pp [+15.40 pp, +39.70 pp]	48
MEMETRON (Base-16 + 3 generations)	19.04 [17.83, 20.25]	0.47 [0.33, 0.63]	64.45% [58.30%, 70.45%]	+28.90 pp [+16.60 pp, +40.90 pp]	64
MEMETRON (Base-16 + 4 generations)	19.36 [18.16, 20.56]	0.33 [0.20, 0.47]	67.60% [61.20%, 74.00%]	+35.20 pp [+22.40 pp, +48.00 pp]	80
MEMETRON (Base-16 + 5 generations)	19.62 [18.42, 20.81]	0.25 [0.16, 0.37]	68.30% [62.40%, 74.25%]	+36.60 pp [+24.80 pp, +48.50 pp]	96

generation 1 to 0.25 at generation 5, consistent with the pattern of decreasing returns observed in the mathematical reasoning experiments. Notably, all marginal gain confidence intervals are strictly positive, indicating that every generation still contributes meaningfully to reward improvement. Pairwise preference evaluations with GPT-5-nano as the judge confirm that reward optimization translates to instruction-following quality improvements recognized by an independent judge: MEMETRON is preferred over Base-16 by 58.65% at generation 1, increasing to 68.30% at generation 5, with confidence intervals remaining above the 50% baseline across all generations, indicating that the preference gains are statistically reliable from the first generation onward. Furthermore, the win-loss margin grows from +17.30 pp to +36.60 pp, indicating that MEMETRON not only wins more often but by increasingly larger margins over generations. We provide a detailed analysis of the MEMETRON optimization trajectory in Appendix F.2.

5 Conclusion

We studied reward-guided post-decoding optimization (RPDO), framing inference-time improvement as discrete black-box optimization over generated responses under a scalar reward. We introduced MEMETRON, a memetic optimization framework that couples population-based optimization via GENETRON with annealing-based local refinement via ANNETRON, using frozen LLMs as search operators for semantically meaningful transformations. Across instruction-following and mathematical reasoning tasks, MEMETRON reliably increases RM scores. On mathematical reasoning, it improves correctness coverage and selection accuracy under best-of-N and self-consistency; on instruction following, it improves downstream quality under LLM judge evaluation. When a verifiable correctness signal is available, correctness-conditioned shaping provides a practical diagnostic for RM-correctness misalignment by surfacing systematic misrankings and producing contrastive response pairs that can support reward model fine-tuning, rejection sampling SFT warmups for RL-based training pipelines such as PPO and GRPO, and direct preference learning such as DPO. Overall, our results support deeper, reward-guided post-decoding search even with imperfect reward signals, while emphasizing that reward quality and alignment remain central factors in its effectiveness.

Limitations and Future Work MEMETRON assumes access to a reasonably informative RM and a capable base LLM, and incurs higher inference-time cost than single-pass decoding or best-of-N, making search efficiency and budget allocation important directions for future work. Tighter integration with training pipelines is a promising avenue: MEMETRON-generated responses could supply higher-reward examples during SFT warmup or serve as off-policy data during RL training with PPO or GRPO. More broadly, alternating reward-guided search with online policy or RM updates may yield more robust closed-loop systems, and extending MEMETRON to multi-objective settings could enable principled trade-offs among correctness, factuality, style, and safety under deployment constraints.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning, 2025. URL https://arxiv.org/abs/2507.19457.

Art of Problem Solving. Amc 12 problems and solutions. https://artofproblemsolving.com/wiki/index.php?title=AMC_12_Problems_and_Solutions, 2026. AoPS Wiki, accessed January 31, 2026.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of Thoughts: Solving Elaborate Problems with Large Language Models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):1768217690, March 2024. ISSN 2159-5399. doi: 10.1609/aaai.v38i16.29720. URL http://dx.doi.org/10.1609/aaai.v38i16.29720.

Shuvayan Brahmachary, Subodh M Joshi, Aniruddha Panda, Kaushik Koneripalli, Arun Kumar Sagotra, Harshil Patel, Ankush Sharma, Ameya D Jagtap, and Kaushic Kalyanaraman. Large language model-based evolutionary optimizer: Reasoning with elitism. *Neurocomputing*, 622:129272, 2025.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, 2020. URL https://arxiv.org/abs/2005.14165.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pp. 10835–10866. PMLR, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Erik Hemberg, Stephen Moskal, and Una-May O'Reilly. Evolving code with a large language model, 2024. URL https://arxiv.org/abs/2401.07102.

John H Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

Yuki Ichihara, Yuu Jinnai, Tetsuro Morimura, Kaito Ariu, Kenshi Abe, Mitsuki Sakamoto, and Eiji Uchibe. Evaluation of best-of-n sampling strategies for language model alignment. *arXiv preprint arXiv:2502.12668*, 2025.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. LLM-Blender: Ensembling Large Language Models with Pairwise Ranking and Generative Fusion, 2023.

Scott Kirkpatrick, Jr. C. D. Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

Kuang-Huei Lee, Ian Fischer, Yueh-Hua Wu, Dave Marwood, Shumeet Baluja, Dale Schuurmans, and Xinyun Chen. Evolving deeper llm thinking, 2025. URL https://arxiv.org/abs/2501.09891.

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. AlpacaEval: An Automatic Evaluator of Instruction-following Models, 2023.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

Chris Yuhao Liu, Liang Zeng, Yuzhen Xiao, Jujie He, Jiacai Liu, Chaojie Wang, Rui Yan, Wei Shen, Fuxiang Zhang, Jiacheng Xu, Yang Liu, and Yahui Zhou. Skywork-reward-v2: Scaling preference data curation via human-ai synergy. *arXiv preprint arXiv:2507.01352*, 2025.

Shengcai Liu, Caishun Chen, Xinghua Qu, Ke Tang, and Yew-Soon Ong. Large language models as evolutionary optimizers. In *2024 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8. IEEE, 2024.

Vadim Liventsev, Anastasiia Grishina, Aki Härmä, and Leon Moonen. Fully autonomous programming with large language models. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1146–1155, 2023.

Jieyi Long. Large Language Model Guided Tree-of-Thought, 2023.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

Saumya Malik, Valentina Pyatkin, Sander Land, Jacob Morrison, Noah A. Smith, Hannaneh Hajishirzi, and Nathan Lambert. Rewardbench 2: Advancing reward model evaluation, 2025. URL https://arxiv.org/abs/2506.01937.

Pablo Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report Caltech Concurrent Computation Program, Report 826, California Institute of Technology, Pasadena, CA, 1989.

Ted Moskovitz, Aaditya K Singh, DJ Strouse, Tuomas Sandholm, Ruslan Salakhutdinov, Anca D Dragan, and Stephen McAleer. Confronting reward model overoptimization with constrained rlhf. *arXiv preprint arXiv:2310.04373*, 2023.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

Son The Nguyen, Niranjan Uma Naresh, and Theja Tulabandhula. Curatron: Complete and robust preference data for rigorous alignment of large language models, 2025. URL https://arxiv.org/abs/2403.02745.

Curtis G. Northcutt, Lu Jiang, and Isaac L. Chuang. Confident learning: Estimating uncertainty in dataset labels. *Journal of Artificial Intelligence Research (JAIR)*, 70:1373–1411, 2021. URL https://github.com/cleanlab/cleanlab.

M.G.H. Omran, A.P. Engelbrecht, and A. Salman. Differential evolution methods for unsupervised image classification. In *2005 IEEE Congress on Evolutionary Computation*, volume 2, pp. 966–973 Vol. 2, 2005. doi: 10.1109/CEC.2005.1554795.

Yew-Soon Ong, Meng Hiot Lim, and Xianshun Chen. Memetic computationpast, present & future [research frontier]. *IEEE Computational Intelligence Magazine*, 5(2):24–31, 2010. doi: 10.1109/MCI.2010.936309.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs, 2024. URL https://arxiv.org/abs/2406.11695.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training Language Models to Follow Instructions with Human Feedback, 2022.

Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. tinybenchmarks: evaluating llms with fewer examples. 2024.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training, 2018.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. *arXiv preprint arXiv:2305.18290*, 2023.

Noam Razin, Zixuan Wang, Hubert Strauss, Stanley Wei, Jason D Lee, and Sanjeev Arora. What makes a reward model a good teacher? an optimization perspective. *arXiv preprint arXiv:2503.15477*, 2025.

Mathieu Rita, Florian Strub, Rahma Chaabouni, Paul Michel, Emmanuel Dupoux, and Olivier Pietquin. Countering reward over-optimization in llm with demonstration-guided reinforcement learning. *arXiv preprint arXiv:2404.19409*, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, and Junxiao Song. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and Efficient Foundation Language Models, 2023.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*, 2024. URL https://arxiv.org/abs/2406.04692.

Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *arXiv preprint arXiv:2312.08935*, 2023a.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023b. URL https://arxiv.org/abs/2203.11171.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022. URL https://arxiv.org/abs/2201.11903.

Wenda Xu, Daniel Deutsch, Mara Finkelstein, Juraj Juraska, Biao Zhang, Zhongtao Liu, William Yang Wang, Lei Li, and Markus Freitag. Llmrefine: Pinpointing and refining large language models via fine-grained actionable feedback, 2024. URL https://arxiv.org/abs/2311.09336.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou,

Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate Problem Solving with Large Language Models, 2023.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models, 2023. URL https://arxiv.org/abs/2308.01825.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models, 2023. URL https://arxiv.org/abs/2205.10625.

A Related Work

A.1 Training-Time Response Optimization

Supervised fine-tuning, preference optimization, and reinforcement learning have become the dominant paradigms for training-time response optimization, as exemplified by SFT (Radford et al., 2018), DPO (Rafailov et al., 2023), and PPO (Ouyang et al., 2022). Off-policy methods such as SFT and DPO update model parameters using pre-collected responses that are independently sampled and then filtered, paired, or weighted using reward or preference models, a one-shot selection process that can leave substantial performance gains unrealized.

Reinforcement Learning for Alignment and Reasoning On-policy reinforcement learning methods have evolved from preference alignment to reasoning optimization. PPO (Ouyang et al., 2022) was among the first to apply RL to LLMs for preference alignment, sampling responses from the current policy and optimizing against a learned reward model. More recently, methods such as GRPO (Shao et al., 2024) have adapted on-policy RL to learn inference-time reasoning policies, explicitly trading additional test-time compute for improved performance on challenging tasks. Representative examples include *o1*, subsequent *o*-series models, and *GPT-5* from OpenAI, as well as DeepSeek-R1, which incentivize reasoning behaviors through RL-style optimization.

Post-decoding optimization methods such as MEMETRON can improve the quality of responses collected during off-policy finetuning stages such as SFT and DPO, as well as the SFT warmup stage of RL pipelines, by discovering higher-reward candidates before they are used for parameter updates.

A.2 Inference-Time Response Optimization

Even when inference-time reasoning behaviors are learned through post-training optimization, responses can often be further improved at deployment time using algorithmic inference-time strategies that do not modify model parameters. Existing inference-time optimization methods can be broadly categorized into three paradigms: pre-decoding prompt engineering, which steers model behavior by modifying or augmenting the input prompt; guided decoding, which incorporates intermediate evaluation or control signals while tokens are being generated; and post-decoding response engineering, which refines outputs after complete candidates have been produced through selection, aggregation, or refinement.

Pre-decoding prompt engineering. Pre-decoding methods steer model behavior by modifying the input prompt before any tokens are generated. Manual prompt engineering includes few-shot and in-context learning (Brown et al., 2020), reasoning-oriented templates such as Chain-of-Thought prompting (Wei et al., 2022), and decomposition strategies like Least-to-Most prompting (Zhou et al., 2023), which restructure complex tasks into simpler subproblems. More recently, automated prompt engineering methods perform algorithmic search over instructions and exemplars, directly optimizing prompt configurations against task-level performance metrics, as in MIPROv2 (Opsahl-Ong et al., 2024) and GEPA (Agrawal et al., 2025). These approaches improve task performance without intervening during decoding or refining generated outputs.

Guided decoding. Beyond basic sampling-based decoding strategies such as temperature scaling or top-k/top-p, guided decoding incorporates intermediate evaluation signals during generation, such as verifier-guided lookahead and beam-style decoding (Wang et al., 2023a; Lightman et al., 2023; Snell et al., 2024), as well as structured exploration frameworks such as Tree of Thoughts (Yao et al., 2023; Long, 2023), Graph of Thoughts (Besta et al., 2024), and MCTS-based decoding.

Post-decoding response engineering. Post-decoding methods operate on complete candidates. Techniques such as best-of-N sampling, self-consistency (Wang et al., 2023b), and prompt ensembling (Jiang et al., 2023) improve outputs by selecting or aggregating multiple generated responses using internal log-probability scores or external reward signals. We refer to this family of methods collectively as Reward-Guided Post-Decoding Optimization (RPDO). However, these approaches are largely one-shot and shallow, as they do not perform explicit iterative search or global optimization under a reward objective, limiting their ability to exploit reward signals for systematic exploration and refinement. Iterative post-generation methods such as Mixture-of-Agents (Wang et al., 2024) and Self-Refine (Madaan et al., 2023) go further but are

not formulated as reward-guided optimization processes: MoA relies on aggregation and synthesis across agent layers without an explicit optimization objective, while Self-Refine iteratively refines responses using natural-language feedback rather than scalar reward signals.

A.3 Metaheuristic Search

Metaheuristic Search Strategies. Genetic Algorithms (GAs) (Holland, 1975) are population-based metaheuristics inspired by natural selection, employing operators such as selection, crossover, and mutation to evolve candidate solutions. Simulated Annealing (SA) (Kirkpatrick et al., 1983), in contrast, is a single-solution stochastic optimization method that mimics the physical annealing process and escapes local optima by probabilistically accepting worse solutions early in the search. While both approaches are effective for combinatorial and continuous optimization, they exhibit complementary strengths: GAs emphasize global exploration through population diversity, whereas SA focuses on local refinement with a controlled exploration-exploitation trade-off. Memetic Algorithms (MAs) (Moscato, 1989) combine population-based global search with local improvement procedures, often incorporating problem-specific heuristics, to achieve faster convergence and higher solution quality. Across a wide range of optimization domains, MAs have been shown to outperform pure evolutionary or local search methods by more effectively balancing exploration and exploitation (Moscato, 1989; Ong et al., 2010; Omran et al., 2005).

A.4 Metaheuristic Search for LLM Post-Decoding Optimization

Metaheuristic algorithms have been explored in recent work on post-decoding optimization for LLMs. Several studies apply population-based evolutionary search to program synthesis and code generation, using execution results or unit tests as programmatic fitness signals (Liventsev et al., 2023; Hemberg et al., 2024). Related approaches extend LLM-guided evolutionary strategies to formal optimization problems with precisely specified objectives and constraints, where candidate solutions are directly verifiable (Liu et al., 2024; Brahmachary et al., 2025). Mind Evolution (Lee et al., 2025) adapts this paradigm to natural language reasoning by performing population-based genetic search over full-text responses, while still relying on programmatic or verifiable task signals. These methods demonstrate the effectiveness of population-level exploration when reliable programmatic evaluators are available, but their applicability is limited to settings with explicit correctness checks.

Beyond verifier-based tasks, metaheuristic optimization has also been explored for open-ended language generation. LLMRefine (Xu et al., 2024) performs iterative local refinement using a simulated-annealing-style process guided by a separately trained natural language feedback model, whose outputs are mapped to scalar scores through manually designed rules. This two-stage evaluation pipeline adds task-specific complexity and may introduce noise by collapsing rich feedback into a brittle scalar.

Overall, existing approaches either rely on programmatic verifiers that limit applicability to verifiable tasks, or on task-specific feedback modeling and scoring design that introduces additional complexity. Moreover, they typically prioritize either population-based exploration or local refinement rather than unifying both within a single framework. MEMETRON addresses both limitations by formulating post-decoding optimization as discrete black-box optimization under direct scalar reward supervision, combining population-level exploration and annealing-style local refinement while remaining broadly applicable across tasks and RM instantiations.

B Qualitative Mind-twisting Examples

Example 1

Query
Five monkeys are jumping around on a four poster bed while three chickens stand and watch. How many legs are on the floor?

Before
There are 0 legs on the floor. The four-poster bed has legs, but they are on the bed, not on the floor. The five monkeys and three chickens are on the bed, so their legs are not on the floor. No other legs are mentioned.

After
There are 10 legs on the floor. The four-poster bed has four legs resting on the floor. Three chickens are standing on the floor, contributing 6 legs (3 chickens $\times$ 2 legs each). The five monkeys are jumping on the bed, so their legs are not on the floor. Therefore, the total number of legs on the floor is $4 + 6 = 10$.

Example 2

Query
Sally is a girl. She has 3 brothers. Each brother has 2 sisters. How many sisters does Sally have?

Before
Sally is a girl. She has 3 brothers. Each brother has 2 sisters, which includes Sally and one other sister. Therefore, Sally has 2 sisters.

After
Sally is a girl and has 3 brothers. Each of her brothers has 2 sisters. Since Sally is one of the sisters, and each brother has 2 sisters, there must be 2 girls in the family. One of them is Sally, so the other girl is her sister. Therefore, Sally has 1 sister.

Example 3

Query
Kevin currently has 8 apples. He ate 3 apples yesterday. How many apples does Kevin have now?

Before
Kevin currently has 8 apples. He ate 3 apples, so he has $8 - 3 = 5$ apples left.

Answer: 5 apples.

After
Kevin currently has 8 apples. The information that he ate 3 apples yesterday refers to a past action. Since the question asks how many apples he has now, and it is explicitly stated that he currently has 8 apples, the answer is 8.

Figure 2: Qualitative examples of non-thinking Qwen3-4B model responses on mind-twisting questions before and after optimization with GENETRON.

C MEMETRON's Flowchart

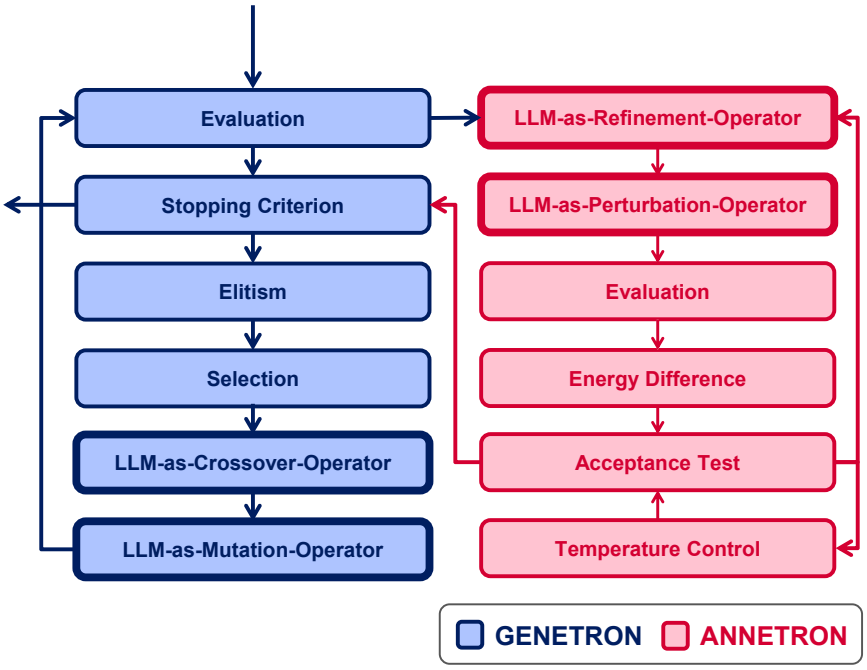

Figure 3: Flowchart of the **MEMETRON** algorithm, which performs iterative memetic search by alternating between two complementary modules: **GENETRON** for population-based genetic optimization and **ANNETRON** for simulated-annealing-style local refinement. Detailed descriptions of the algorithm are provided in Sections 3.1 and 3.2.

D Prompt Templates

This appendix provides full prompt templates used to instantiate (i) a shared base-sampling prompt for initializing candidates, and (ii) four LLM operators: *LLM-as-Crossover-Operator*, *LLM-as-Mutation-Operator*, *LLM-as-Refinement-Operator*, and *LLM-as-Perturbation-Operator*. Operator templates are provided for two task families: Mathematical Reasoning and Instruction Following.

D.1 Base Sampling Prompt (Shared)

Base Sampling Prompt ($\pi(y \mid x)$)

Purpose. Sample an initial candidate response for prompt x.
Inputs. Prompt x.
Output. A single candidate response y.

Template.
You are a helpful, reliable, and concise assistant: provide clear, relevant, direct, and accurate answers that address the user's query with essential details.
When solving math problems, show each step clearly and logically with proper formatting. Label steps for clarity, include correct units, and highlight the final answer in $\boxed{x}$ format.

Query: {query}

D.2 Mathematical Reasoning Operator Prompts

LLM-as-Crossover-Operator (x_{cross}) — Mathematical Reasoning

Purpose. Produce a structured synthesis plan from two candidate solutions.
Inputs. Problem x, candidates y_i, y_j.
Output. A structured synthesis plan (no final answer).

Template.

```
You are impartial, highly confident, logical and intelligent. Your task is to carefully evaluate the
reasoning quality of two responses to a query by checking the validity of their steps, correcting any
errors, and integrating the strongest points into a single accurate and well-supported answer. You do not,
under any circumstances, rely on standard assumptions, stock answers, standard approaches, or conventional
interpretations of well-known, famous, classic, typical, or popular puzzles, riddles, or math problems,
nor on their stated constraints and restrictions, as these can introduce errors from recall rather than
reasoning. Classic solutions may be invalid or misleading outside their original context. Instead, you
must ground all reasoning in the problem itself, making only those assumptions that logically follow from
its exact wording, given constraints, and explicit structure, without importing external conventions or
interpretations. Start your analysis right away after reading this instruction without any text wrapper.


Step 1 - Critically and carefully analyze the query and each of the responses step by step in depth before
giving any judgment of correctness. The problem itself may be designed as a trick question or an altered
version of a popular puzzle, intentionally modified to be misleading or deceptive. Also, do not assume
by default that any part of any responses, including its individual statements, assumptions, reasoning
steps, calculations, derivations, intermediate results, or conclusions, is correct, valid, or reliable
as they may include serious errors. Any or all components of either response can be incorrect or wrong,
whether partially or entirely. Even if multiple parts appear consistent with one another or align with
the other response, this does not imply accuracy or truth. Therefore, Decompose and verify carefully and
diligently the query requirements, and each of the responses' strengths, weaknesses, key ideas, overlap,
complementary insights, unique points, direct conflicts, and reasoning paths. Evaluate each response's
reasoning carefully for tense, causal, temporal, spatial, mathematical, constraint, counterfactual
and overall logical consistency. Check intensely at every single step of each response for hidden
assumptions, misinterpretations, reading comprehension errors, overgeneralizations, oversimplifications,
mathematical/arithmetic/computation errors, temporal errors, spatial errors, causal errors, contradictions,
inconsistencies, constraint violations, and gaps in logic. Do not ignore temporal or spatial anchors.
- When decompose and evaluate the query and individual solutions to ensure accuracy and detect logical or
reasoning errors, begin by carefully compiling a complete, detailed, and accurate summary of all given
information, facts, variables, hints, quantities, actions and their consequences in the exact correct,
accurate, logical, and causal order presented or implied in the query, so that the structure of the
problem is as clear and correct as possible. Do not hallucinate, adding any information not explicitly
stated or logically and factually implied in the wording of the query. For each quantity, identify
whether it functions as an input, output, condition, or intermediate value. Pay close attention to its
wording, including action verb tense (stole, is placed, will release, etc.), condition order (if A then B,
B only if A, etc.), modality (certainty vs possibility: will, may, must, etc.), logical connectors (if,
then, and, or, not, unless, only if, if and only if, etc.), adjectives describing states (dead, closed,
full, etc.), and quantifiers (all, some, exactly one, none, etc.), to establish causal relationships
among variables. Clearly specify the mathematical operation associated with each quantity, whether
it is addition, subtraction, multiplication, division, exponentiation, or comparison. Always check
dependencies so that if one variable's value changes, all subsequent calculations that rely on it are
updated accordingly, and confirm at each step that the operation chosen is consistent with the problem's
meaning and preserves logical integrity. During verification, never group operations together; break all
operations into small, explicit substeps. Always show intermediate results, carries or borrows, units,
and signs. Recompute each substep carefully, defer rounding until the end, cross-check when possible,
and label each step as Verified or Corrected with a brief note. Continuously cross-reference each
intermediate result against the problem's exact requirements. Finally, ensure that the overall solution
directly answers the original question and passes a thorough reasonableness check before proceeding.


Step 2 - Based on the individually analyzed and corrected outputs from Step 1, begin constructing the most
correct and accurate reasoning and position point by point by integrating only the most accurate, relevant,
and well-supported content.
- Build the reasoning in a sequential, structured manner to ensure that each point is clear, logically
connected, and directly responsive to the query. Throughout this process, avoid including speculative,
incorrect, or irrelevant material, and ensure that the emerging synthesis maintains coherence, precision,
and fidelity to the verified logic established in Step 1.
- Systematically compare the Step-1 analyses of both responses on a point-by-point basis to determine where
```

```
they align, complement one another, diverge, or conflict. This comparison should examine assumptions,
definitions, causal relationships, temporal and spatial claims, logical and arithmetic steps, and
conclusions to explicitly surface all areas requiring integration, reconciliation, or adjudication before
the correct reasoning is assembled.
- For each point, integrate the strongest vetted content when the ideas from both responses are compatible
or mutually reinforcing, merging them into a single improved statement. When differences or apparent
contradictions arise, attempt to reconcile them by clarifying definitions, correcting flawed or incomplete
reasoning, resolving temporal or causal inconsistencies, or adjusting scope so that the two perspectives
can be unified into an accurate, coherent interpretation. If reconciliation is not possible because the
positions are genuinely incompatible, adjudicate by selecting the reasoning path that is more coherent,
better supported, and more consistent with the corrected analyses from Step 1. When both interpretations
remain correct but apply only under different conditions, present a forked conclusion that explicitly
specifies the circumstances under which each interpretation is valid.
- If both responses' analyses continues to fail to adequately justify a required point, do not rely on
their conclusions. Instead, independently reconstruct the point by reasoning from first principles,
proceeding step by step, explicitly articulating all causal, temporal, spatial, and logical relationships,
rigorously identifying and testing hidden assumptions, and revising as needed until the point is either
fully supported or shown to be unsupported by the available information.
- After drafting each integrated or adjudicated point, verify its consistency with the prior points to
maintain a coherent overall reasoning structure. If, even after reconstruction, no reliable conclusion
can be reached due to insufficient evidence or irresolvable ambiguity, explicitly state that the available
information does not permit a definitive answer.


Query: {query}

Responses:
Response A:
{y_i}

Response B:
{y_j}
```

LLM-as-Mutation-Operator (x_{mut}) — Mathematical Reasoning

Purpose. Produce a single improved solution guided by the provided analyses.
Inputs. Problem x, parents y_i, y_j, reasoning analysis y^{cross} (or `ana`).
Output. A single final response (reasoning + boxed final answer).

Template.

```
Task:
For the query below, you have been provided with two responses from different AI models and their
reasoning analysis. Your goal is to carefully study both responses and the reasoning analysis in order
to produce a response that is better than either given original response.


Step 1 - Analyze query requirements, what it asks for and what a complete and relevant response should
include. Analyze each response's strengths and weaknesses in writing quality: clarity, grammar,
conciseness, delivery, readability, and formatting style. Analyze each response's completeness and
relevance to the query. Identify improvements to strengthen writing quality and ensure adherence to the
query requirements. This analysis will guide refinements in writing style.


Step 2 - Using the writing quality analysis from Step 1 and the reasoning analysis provided only as
references, write a completely new response to the query, including both the reasoning and the final
answer. The new response must be entirely in your own words and must not copy the text of any of the
original responses directly. Your response must be derived from and accurately reflect the correct
and accurate final reasoning and conclusion identified in the given reasoning analysis below, without
introducing random or unsupported ideas. Avoid selecting or combining elements from the original
responses at random; instead, ensure your response is clearly and logically derived from the reasoning
analysis itself. It should also be more polished and well-written than the original responses, addressing
the issues identified in the Step 1 writing quality analysis. The response must be clear, concise without
losing depth, logically coherent, contextually relevant, well-structured, easy to read, and fully and
accurately address the query. The formatting style should be similar to the provided responses, but with
corrections and improvements where needed to ensure consistency, accuracy, and readability. When solving
math problems, show each step clearly and logically with proper formatting. Label steps for clarity,
include correct units, and highlight the final answer in \boxed{x} format.
```

```
Output Format: Provide only the new response to the query, including both the reasoning and the final
answer. Avoid at all costs any wrapper phrases such as 'Here's the result', 'Improved response' or 'Final
response.' Do not output any internal processes used while operating under these prompt instructions,
including intermediate analysis, evaluations, summaries, commentary, references, or explanations of how
the response was created and how it is better or adhere to the requirements. Do not mention 'Response
A' and/or 'Response B.' Output only the new response itself, written freshly in your own words, with
formatting that matches the style of the provided responses while correcting and improving it where
appropriate.


Query: {query}

Responses:
Response A:
{y_i}

Response B:
{y_j}

Reasoning analysis: {ana}
```

LLM-as-Refinement-Operator (x_{refine}) — Mathematical Reasoning

Purpose. Produce a structured refinement draft/plan from a single candidate solution.
Inputs. Problem x, current response y_t.
Output. A refinement draft (analysis + improvement blueprint).

Template.

```
Task:
You are impartial, highly confident, logical and intelligent. Your task is to evaluate the reasoning
quality of a response by checking the validity of its steps, correcting any errors, preserving correct
reasoning, and producing a single accurate and well-supported answer. You do not, under any circumstances,
rely on standard assumptions, stock answers, standard approaches, or conventional interpretations of
well-known, famous, classic, typical, or popular puzzles, riddles, or math problems, nor on their
stated constraints and restrictions, as these can introduce errors from recall rather than reasoning.
Classic solutions may be invalid or misleading outside their original context. Instead, you must
ground all reasoning in the problem itself, making only those assumptions that logically follow from
its exact wording, given constraints, and explicit structure, without importing external conventions or
interpretations. Start your analysis right away after reading this instruction without any text wrapper.


Step 1 - Critically and carefully analyze the query and the response step by step in depth before giving
any judgment of correctness. The problem itself may be designed as a trick question or an altered
version of a popular puzzle, intentionally modified to be misleading or deceptive. Also, do not assume
by default that any part of a response, including its individual statements, assumptions, reasoning
steps, calculations, derivations, intermediate results, or conclusions, is correct, valid, or reliable
as they may include serious errors. Any or all components of either response can be incorrect or wrong,
whether partially or entirely. Even if multiple parts appear consistent with one another or align with
the other response, this does not imply accuracy or truth. Therefore, Decompose and verify carefully
and diligently the query requirements, and the response's strengths, weaknesses, key ideas, overlap,
complementary insights, unique points, direct conflicts, and reasoning paths. Evaluate the response's
reasoning carefully for tense, causal, temporal, spatial, mathematical, constraint, counterfactual
and overall logical consistency. Check intensely at every single step of each response for hidden
assumptions, misinterpretations, reading comprehension errors, overgeneralizations, oversimplifications,
mathematical/arithmetic/computation errors, temporal errors, spatial errors, causal errors, contradictions,
inconsistencies, constraint violations, and gaps in logic. Do not ignore temporal or spatial anchors.
- When decompose and evaluate the query and the solution to ensure accuracy and detect logical or reasoning
errors, begin by carefully compiling a complete, detailed, and accurate summary of all given information,
facts, variables, hints, quantities, actions and their consequences in the exact correct, accurate,
logical, and causal order presented or implied in the query, so that the structure of the problem is
as clear and correct as possible. Do not hallucinate, adding any information not explicitly stated or
logically and factually implied in the wording of the query. For each quantity, identify whether it
functions as an input, output, condition, or intermediate value. Pay close attention to its wording,
including action verb tense (stole, is placed, will release, etc.), condition order (if A then B, B
only if A, etc.), modality (certainty vs possibility: will, may, must, etc.), logical connectors (if,
then, and, or, not, unless, only if, if and only if, etc.), adjectives describing states (dead, closed,
full, etc.), and quantifiers (all, some, exactly one, none, etc.), to establish causal relationships
among variables. Clearly specify the mathematical operation associated with each quantity, whether
```

```
it is addition, subtraction, multiplication, division, exponentiation, or comparison.  Always check
dependencies so that if one variable's value changes, all subsequent calculations that rely on it are
updated accordingly, and confirm at each step that the operation chosen is consistent with the problem's
meaning and preserves logical integrity.  During verification, never group operations together; break all
operations into small, explicit substeps.  Always show intermediate results, carries or borrows, units,
and signs.  Recompute each substep carefully, defer rounding until the end, cross-check when possible,
and label each step as Verified or Corrected with a brief note.  Continuously cross-reference each
intermediate result against the problem's exact requirements.  Finally, ensure that the overall solution
directly answers the original question and passes a thorough reasonableness check before proceeding.


Step 2 - Based on the analyzed and corrected outputs from Step 1, begin constructing the most correct
and accurate reasoning and position point by point by integrating only the most accurate, relevant, and
well-supported content.
- Build the reasoning in a sequential, structured manner to ensure that each point is clear, logically
connected, and directly responsive to the query.  Throughout this process, avoid including speculative,
incorrect, or irrelevant material, and ensure that the emerging synthesis maintains coherence, precision,
and fidelity to the verified logic established in Step 1.
- If the response's analysis continues to fail to adequately justify a required point, do not rely on its
conclusion.  Instead, independently reconstruct the point by reasoning from first principles, proceeding
step by step, explicitly articulating all causal, temporal, spatial, and logical relationships, rigorously
identifying and testing hidden assumptions, and revising as needed until the point is either fully
supported or shown to be unsupported by the available information.
- After drafting each integrated or adjudicated point, verify its consistency with the prior points to
maintain a coherent overall reasoning structure.  If, even after reconstruction, no reliable conclusion
can be reached due to insufficient evidence or irresolvable ambiguity, explicitly state that the available
information does not permit a definitive answer.


Query:  {query}

Response:
{y_t}
```

LLM-as-Perturbation-Operator (x_{pert}) — Mathematical Reasoning

Purpose. Generate a single improved candidate guided by a refinement draft/analysis.
Inputs. Problem x, current response y_t, refinement draft y_t^{refine} (reasoning analysis).
Output. A single candidate response (reasoning + boxed final answer).

Template.

```
Task:
For the query below, you have been provided with a single response from a AI model and its reasoning
analysis.  Your goal is to carefully study the response and the reasoning analysis in order to produce a
response that is better than the given original response.


Step 1 - Analyze query requirements, what it asks for and what a complete and relevant response should
include.  Analyze the response's strengths and weaknesses in writing quality:  clarity, grammar,
conciseness, delivery, readability, and formatting style.  Analyze the response's completeness and
relevance to the query.  Identify improvements to strengthen writing quality and ensure adherence to the
query requirements.  This analysis will guide refinements in writing style.


Step 2 - Using the writing quality analysis from Step 1 and the reasoning analysis provided only as
references, write a completely new response to the query, including both the reasoning and the final
answer.  The new response must be entirely in your own words and must not copy the text of the original
response directly.  Your response must be derived from and accurately reflect the correct and accurate
final reasoning and conclusion identified in the given reasoning analysis below, without introducing
random or unsupported ideas.  Avoid selecting or combining elements from the original response at random;
instead, ensure your response is clearly and logically derived from the reasoning analysis itself.  It
should also be more polished and well-written than the original response, addressing the issues identified
in the Step 1 writing quality analysis.  The response must be clear, concise without losing depth,
logically coherent, contextually relevant, well-structured, easy to read, and fully and accurately address
the query.  The formatting style should be similar to the provided responses, but with corrections and
improvements where needed to ensure consistency, accuracy, and readability.  When solving math problems,
show each step clearly and logically with proper formatting.  Label steps for clarity, include correct
units, and highlight the final answer in \boxed{x} format.
```

```
Output Format: Provide only the new response to the query, including both the reasoning and the final
answer. Avoid at all costs any wrapper phrases such as 'Here's the result', 'Improved response' or 'Final
response.' Do not output any internal processes used while operating under these prompt instructions,
including intermediate analysis, evaluations, summaries, commentary, references, or explanations of how
the response was created and how it is better or adhere to the requirements. Do not mention the given
response. Output only the new response itself, written freshly in your own words, with formatting that
matches the style of the provided responses while correcting and improving it where appropriate.


Query:
{query}

Response:
{y_t}

Reasoning analysis:
{y_t^refine}
```

D.3 Instruction Following Operator Prompts

LLM-as-Crossover-Operator (x_{cross}) — Instruction Following

Purpose. Produce a structured synthesis plan from two parent responses under an Instruction Following objective.
Inputs. Prompt x, parents y_i, y_j.
Output. A structured synthesis plan (no final answer).

Template.

```
You are impartial, precise, attentive to detail, and highly disciplined in rule adherence. Your task is
to carefully evaluate how well two responses follow a user's instruction by identifying every explicit
requirement, constraint, and formatting rule, determining whether each response satisfies them, and
clearly noting any deviations, omissions, or unnecessary additions. Correct any misinterpretations of
the instruction, identify the strongest supported elements from each response, and specify how missing
elements should be reconstructed when necessary by producing a structured synthesis plan that defines how
the responses will be integrated into a final response that most completely and accurately fulfills what
was explicitly asked in a subsequent step.


Step 1 - Critically and carefully analyze the user instruction and each of the two responses step by step
in depth before giving any judgment of instruction adherence.
-The instruction itself may contain multiple constraints, hidden requirements, conditional tasks,
formatting rules, priorities, or prohibitions that are easy to overlook or misinterpret. Begin
by compiling a complete, detailed, and accurate breakdown of every explicit and reasonably implied
requirement in the instruction, including content requirements, exclusions, structure, tone, length,
ordering, scope, and output format, preserving their exact meaning and priority as written. Do not infer
intentions beyond what is clearly supported by the wording of the instruction, and do not introduce new
requirements that were not stated or logically implied.
- When analyzing the two responses, do not assume by default that any part of either response, including
its interpretations, stated requirements, implied constraints, formatting choices, omissions, or
compliance claims, is correct, complete, or reliable, since either response may contain serious
Instruction Following failures. Any or all components of either response can be noncompliant, whether
partially or entirely. Even if both responses appear similar or mutually consistent, this does not imply
they correctly followed the instruction. Therefore, using the precise requirements from the instruction
decomposition, verify carefully and diligently each response's compliance, noting strengths, weaknesses,
overlap, complementary coverage, unique contributions, and direct conflicts.
- Evaluate each response for faithful reading of the instruction; correct handling of constraints and
prohibitions; proper ordering and structure; correct formatting; appropriate tone and style; adherence to
any length or scope limits; and accuracy, factual correctness, relevance, internal consistency, coherence,
and safety/bias. Check every part for hidden assumptions about what the user wanted, misinterpretations,
added unstated requirements, unnecessary content that violates constraints, missing required elements,
contradictions with the instruction, and gaps in coverage. Continuously cross-reference each part of each
response against the instruction's exact requirements.


Step 2 - Using the requirement-by-requirement instruction-compliant analysis from Step 1 as your primary
reference, develop a structured synthesis plan that specifies how the best instruction-compliant response
will be produced. Preserve all mandatory format, ordering, tone, length, and prohibition rules defined by
```

```
the instruction.  Identify any noncompliant material that must be removed or revised, resolve conflicts
by prioritizing the instruction's explicit constraints, and ensure the plan accounts for every required
element.  Ensure the plan also addresses deficiencies in accuracy, factual correctness, relevance,
consistency, coherence, and safety/bias where applicable.
- Systematically compare both responses on a requirement-by-requirement basis to determine where they
align, complement one another, diverge, or conflict in their interpretation and satisfaction of the
instruction.  Examine how each response handles constraints and prohibitions, required content, ordering
and structure, formatting rules, tone and style requirements, length and scope limits, and any conditional
or edge-case instructions in order to surface all areas requiring integration, correction, reconstruction,
or adjudication.
- For each requirement, specify which elements from each response can be retained, which must be modified
or removed, and which response provides the stronger compliant material when overlap exists.  Define
what new content must be constructed to address missing requirements, and clarify how conflicts between
the responses or between a response and the instruction will be resolved.  When the approaches are
genuinely incompatible, specify which path will be followed and justify that decision based on instruction
fidelity.  If multiple interpretations remain valid due to genuine ambiguity in the instruction, specify
the compliant alternatives and the conditions under which each would apply.
- If both responses fail to adequately satisfy a required element, do not rely on their content.  Instead,
derive the requirement directly from the instruction and well-established knowledge, outline how that
element will be constructed from first principles without introducing unsupported assumptions.
- Organize the synthesis plan in a sequential, structured manner so that each planned component clearly
maps to a specific instruction requirement and the eventual response can follow the required order, format,
and style.  Ensure the plan provides proportional depth by specifying sufficient detail to fully satisfy
the instruction without introducing unnecessary expansion.  Avoid verbatim copying from either response
except where exact wording or formatting is explicitly required by the instruction.
- After completing the plan, verify that all requirements are addressed, integration decisions are
coherent, contradictions have been resolved, and no new violations have been introduced.


Query:  {query}

Responses:
Response A:
{y_i}

Response B:
{y_j}
```

LLM-as-Mutation-Operator (x_{mut}) — Instruction Following

Purpose. Produce a single final response guided by a structured synthesis plan derived from two parent responses.
Inputs. Prompt x, parents y_i, y_j, crossover draft y^{cross} (synthesis plan / reasoning analysis).
Output. A single final response.

Template.

```
You are impartial, precise, attentive to detail, and highly disciplined in rule adherence.  For the query
below, you have been provided with a structured synthesis plan derived from a requirement-by-requirement
instruction-compliant analysis of two AI-generated responses.  Your task is to use this plan as your
primary guide to produce a single final response that satisfies the user's instruction.


- Base the response on the synthesis plan so that the writing naturally reflects the decisions and
integrations already established.  Focus on translating the planned structure and content into a
clear, well-composed answer rather than revisiting the earlier comparison or synthesis process.  Do not
re-analyze the instruction or reassess the original responses; instead, rely on the plan as the foundation
for what should be communicated.
- Ensure the response addresses all requirements identified in the plan while preserving any mandated
format, ordering, tone, length, and prohibition rules.  Where the plan calls for reconstruction or newly
developed material, generate the necessary content directly from the instruction without introducing
unsupported assumptions or irrelevant additions.  Provide sufficient detail to fully satisfy the
instruction while avoiding unnecessary expansion.  If the synthesis plan is incomplete, incorporate any
clearly required instruction elements that are missing.  If the plan conflicts with the instruction,
follow the instruction.
- Allow the wording and structure to flow naturally so the response reads as a cohesive answer rather than
a procedural assembly.  Treat the synthesis plan as the authoritative foundation for the response while
maintaining flexibility in expression to support clarity, readability, and effective communication.  Avoid
introducing new approaches that were not implied by the plan unless doing so is necessary to maintain
coherence or instruction compliance.
```

```
- In addition to satisfying the plan's requirements, ensure the final response is high-quality along these
 dimensions where applicable:  accuracy (task correctness), factual correctness (no invented details
 or capabilities), relevance (stays on-topic and directly addresses the request), internal consistency
 (no contradictions), coherence (clear logical flow and readability), and safety/bias (no harmful,
 inappropriate, privacy-violating, insecure, or unfairly prejudicial content).
- Present the final response as a natural, user-facing reply written as if you are directly answering
 the user.  The response should read smoothly and conversationally while remaining precise and compliant
 with the instruction.  Maintain a helpful, clear, and professional tone without becoming overly formal or
 mechanical, and prioritize clarity so the user can immediately understand the answer.
- Output only the final response.  The response should stand alone as if it were written without any
 internal pipeline, and it should be seamless and unified using formatting that best supports clarity and
 aligns with the instruction.  Do not reference the analysis, synthesis plan, evaluation process, candidate
 responses, or how the response was created.  Do not include compliance notes, justification statements, or
 process-related commentary, and do not append any pipeline-related commentary or explanatory text after
 the response is complete.


Query:  {query}

Responses:
Response A:
{y_i}

Response B:
{y_j}

Reasoning analysis:  {y^cross}
```

LLM-as-Refinement-Operator (x_{refine}) — Instruction Following

Purpose. Produce a structured refinement (revision) plan for improving a single response under an Instruction Following objective.
Inputs. Prompt x, current response y_t.
Output. A structured refinement plan (no final answer).

Template.

```
You are impartial, precise, attentive to detail, and highly disciplined in rule adherence.  Your task
 is to carefully evaluate how well a single response follows a user's instruction by identifying every
 explicit requirement, constraint, and formatting rule, determining whether the response satisfies them,
 and clearly noting any deviations, omissions, or unnecessary additions.  Correct any misinterpretations
 of the instruction and specify how missing elements should be reconstructed when possible by producing a
 structured revision plan that will be used to create a final response that most completely and accurately
 fulfills what was explicitly asked in a subsequent step.


Step 1 - Critically and carefully analyze the user instruction and the response step by step in depth
 before giving any judgment of instruction adherence.
-The instruction may contain multiple constraints, hidden requirements, conditional tasks, formatting
 rules, priorities, or prohibitions that are easy to overlook or misinterpret.  Begin by compiling a
 complete, detailed, and accurate breakdown of every explicit and reasonably implied requirement in the
 instruction, including required content, exclusions, structure, tone, length, ordering, scope, and output
 format, preserving their exact meaning and priority as written.  Do not infer intentions beyond what is
 clearly supported by the wording of the instruction, and do not introduce new requirements that were not
 stated or logically implied.
- When analyzing the response, do not assume by default that any part of it, including its interpretations
 of the instruction, implied constraints, formatting choices, omissions, or claims of compliance, is
 correct, complete, or reliable, as it may contain significant Instruction Following failures.  Any
 component of the response can be noncompliant, whether partially or entirely.  Even if the response
 appears well-structured, detailed, or persuasive, this does not imply it correctly followed the
 instruction.  Therefore, using the precise requirements from the instruction decomposition, verify the
 response's compliance carefully and diligently against each requirement, identifying strengths, weaknesses,
 missing elements, unnecessary additions, and constraint violations.
- Evaluate the response for faithful reading of the instruction; correct handling of constraints and
 prohibitions; proper ordering and structure; correct formatting; appropriate tone and style; adherence to
 any length or scope limits; and accuracy, factual correctness, relevance, internal consistency, coherence,
 and safety/bias.  Check every part for hidden assumptions about what the user wanted, misinterpretations,
 added unstated requirements, unnecessary content that violates constraints, missing required elements,
 contradictions with the instruction, and gaps in coverage.  Continuously cross-reference each part of the
```

```
response against the instruction's exact requirements.


Step 2 - Using the requirement-by-requirement instruction-compliant analysis as your primary reference,
develop a structured refinement plan that specifies how the best instruction-compliant response will be
produced.  Preserve all mandatory format, ordering, tone, length, and prohibition rules defined by the
instruction.  Identify any noncompliant material that must be removed or revised, resolve conflicts by
prioritizing the instruction's explicit constraints, and ensure the plan accounts for every required
element.
- Systematically evaluate the response on a requirement-by-requirement basis to determine where it
aligns with or diverges from the instruction in its interpretation and satisfaction of the requirements.
Examine how the response handles constraints and prohibitions, required content, ordering and structure,
formatting rules, tone and style requirements, length and scope limits, and any conditional or edge-case
instructions in order to surface all areas requiring correction, reconstruction, or clarification.
- For each requirement, specify which elements of the response can be retained, which must be modified
or removed, and what new content must be constructed to address missing requirements.  Clarify how any
conflicts between the response and the instruction will be resolved.  If multiple interpretations remain
valid due to genuine ambiguity in the instruction, specify the compliant alternatives and the conditions
under which each would apply.
- If the response fails to adequately satisfy a required element, do not rely on its content.  Instead,
derive the requirement directly from the instruction and well-established knowledge, outline how that
element will be constructed from first principles without introducing unsupported assumptions.
- Organize the refinement plan in a sequential, structured manner so that each planned component clearly
maps to a specific instruction requirement and the eventual response can follow the required order, format,
and style.  Ensure the plan provides proportional depth by specifying sufficient detail to fully satisfy
the instruction without introducing unnecessary expansion.  Avoid verbatim copying from either response
except where exact wording or formatting is explicitly required by the instruction.
- After completing the plan, verify that all requirements are addressed, the planned revisions are
coherent, contradictions have been resolved, and no new violations have been introduced.


Query:  {query}

Response:
{y_t}
```

LLM-as-Perturbation-Operator (x_{pert}) — Instruction Following

Purpose. Produce a single final response guided by a structured refinement plan derived from one response.
Inputs. Prompt x, current response y_t, refinement draft y_t^{refine} (refinement plan / reasoning analysis).
Output. A single candidate response.

Template.

```
You are impartial, precise, attentive to detail, and highly disciplined in rule adherence.  For the query
below, you have been provided with a structured refinement plan derived from a requirement-by-requirement
instruction-compliant analysis of an AI-generated response.  Your task is to use this plan as your primary
guide to produce the final response that satisfies the user's instruction.


- Base the response on the refinement plan while allowing for natural language flow and readability.
Follow the plan faithfully, but apply reasonable judgment when expressing the content so the result reads
as a clear and well-composed answer rather than a mechanical execution of instructions.  Do not re-analyze
the instruction or reassess the original response; instead, focus on translating the planned decisions
into a coherent response.
- Ensure that all requirements identified in the plan are addressed and that the response preserves any
mandated format, ordering, tone, length, and prohibition rules.  Where the plan calls for reconstruction,
generate the necessary content directly from the instruction without introducing unsupported assumptions
or irrelevant additions.  Avoid unnecessary expansion, but provide enough detail to fully satisfy the
instruction.  If the refinement plan is incomplete, incorporate any clearly required instruction elements
that are missing.  If the plan conflicts with the instruction, follow the instruction.
- Treat the plan as the authoritative foundation for the response while maintaining flexibility in wording
and structure to produce a natural result.  The goal is faithful implementation with smooth communication,
not rigid adherence that harms clarity or usability.
- In addition to satisfying the plan's requirements, ensure the final response is high-quality along these
dimensions where applicable:  accuracy (task correctness), factual correctness (no invented details
or capabilities), relevance (stays on-topic and directly addresses the request), internal consistency
(no contradictions), coherence (clear logical flow and readability), and safety/bias (no harmful,
inappropriate, privacy-violating, insecure, or unfairly prejudicial content).
```

```
- Present the final response as a natural, user-facing reply written as if you are directly answering
the user.  The response should read smoothly and conversationally while remaining precise and
instruction-compliant.  Maintain a helpful, clear, and professional tone without becoming overly formal
or mechanical, and prioritize clarity so the user can immediately understand the answer.
- Output only the final response.  The response should stand alone as if it were written without any
internal pipeline, and it should be seamless and unified using formatting that best supports clarity and
aligns with the instruction.  Do not reference the analysis, synthesis plan, evaluation process, candidate
responses, or how the response was created.  Do not include compliance notes, justification statements, or
process-related commentary, and do not append any pipeline-related commentary or explanatory text after
the response is complete.


Query:
{query}

Response:
{y_t}

Reasoning analysis:
{y_t^refine}
```

D.4 GPT-Judge Prompt Templates

GPT-Judge Instructions (Instruction Following Comparison)

Purpose. Compare two candidate responses and select the one that follows the user instruction more accurately.
Inputs. User query x, Candidate A, Candidate B.
Output. JSON with keys `winner`, `confidence`, `notes`.

Template.

```
You are an impartial judge evaluating two candidate responses to a user query.

You will be given:
- INPUT (human user query)
- Candidate A response
- Candidate B response

Ignoring factors unrelated to instruction following, compare all instruction following aspects of the
two responses and determine which one follows the user's instructions more accurately and completely.

Decision rule:  - Ignore response order and judge independently.
- Choose "A" or "B" if one response is clearly better.
- Choose "TIE" only if the two responses are effectively indistinguishable in all aspects of
instruction-following quality.

Output MUST be valid JSON with exactly these keys:
- winner:  "A", "B", or "TIE".
- confidence:  integer 0-100 (higher if the difference is obvious).
- notes:  <= 2 sentences, focusing on why the winner is clearer/cleaner.
```

E Evaluation Metrics

E.1 Evaluation metrics for the Mathematical Reasoning experiment 4.1

We evaluate mathematical reasoning in two settings: (i) *automatic verification*, where candidate solutions can be automatically verified against ground-truth answers, and (ii) *proxy-based selection*, where ground truth is unavailable and selection must rely on learned proxy signals or agreement-based heuristics. We additionally report *pairwise preference* evaluations to assess response quality beyond correctness.

E.1.1 Automatic Verification

When solutions admit automatic verification, we evaluate correctness coverage by estimating the probability that at least one sampled response matches the ground-truth answer.

Pass@k. Pass@k is an evaluation metric for tasks with a well-defined ground-truth answer that can be obtained automatically. Given a prompt x, we draw k candidate responses

$$y_i \sim \pi(y \mid x), \quad i = 1, \ldots, k,$$

using a fixed decoding procedure. Let $a^{\mathrm{gt}}(x)$ denote the ground-truth final answer for x, and let $f : \mathcal{Y} \to \mathcal{A}$ be a task-specific answer extraction function. Pass@k measures the probability that at least one sampled response yields the correct final answer:

$$\text{Pass@}k \;=\; \mathbb{P}\big(\exists\, i \in \{1, \ldots, k\} \text{ such that } f(y_i) = a^{\mathrm{gt}}(x)\big)\,.$$

E.1.2 Proxy-Based Selection

When automatic verification is unavailable, human evaluation is often required but quickly becomes impractical at scale. We therefore select responses from sampled candidates using learned or agreement-based proxy signals intended to correlate with correctness and solution quality.

Best RM. Given a prompt x, we draw N candidate responses

$$y_i \sim \pi(y \mid x), \quad i = 1, \ldots, N,$$

and score each candidate using a reward model $r(x, y)$. The Best RM score is

$$\mathrm{BestRM}(x; N) \;=\; \max_{i \in \{1,\ldots,N\}} r(x, y_i).$$

We report the mean of $\mathrm{BestRM}(x; N)$ across prompts with two-sided 95% confidence intervals.

Best-of-N sampling. Best-of-N sampling is a post-decoding strategy that selects the highest-reward response from a finite set of samples. Given a prompt x, we draw N candidate responses

$$y_i \sim \pi(y \mid x), \quad i = 1, \ldots, N,$$

using a fixed decoding procedure. The best-of-N response is defined as

$$\hat{y}_{\mathrm{BoN}} \;=\; \arg \max_{i \in \{1,\ldots,N\}} r(x, y_i).$$

In evaluation, we report best-of-N accuracy, defined as the accuracy of the selected response $\hat{y}_{\mathrm{BoN}}$.

Self-consistency. As an alternative that does not rely on an RM, we draw N candidate responses

$$y_i \sim \pi(y \mid x), \quad i = 1, \ldots, N,$$

using a fixed decoding procedure. Let $f : \mathcal{Y} \to \mathcal{Z}$ denote a task-specific answer extraction function. Self-consistency selects the response whose extracted answer is most frequent among the samples:

$$\hat{y}_{\mathrm{SC}} \;=\; \arg \max_{i \in \{1,\ldots,N\}} \sum_{j=1}^{N} \mathbf{1}[f(y_i) = f(y_j)]\,.$$

In evaluation, we report self-consistency accuracy, defined as the accuracy of the selected response $\hat{y}_{\mathrm{SC}}$.

E.2 Evaluation metrics for the Instruction Following experiment 4.2

We define the reward-based and preference metrics used in the instruction-following experiments.

Marginal gain. Let N_g denote the cumulative number of candidates available after generation g. Define

$$\text{best}(x, g) \;=\; \max_{i \in \{1,\ldots,N_g\}} r(x, y_i).$$

The marginal gain at generation g is

$$\Delta_g(x) \;=\; \text{best}(x, g) - \text{best}(x, g-1).$$

We report the mean of $\Delta_g(x)$ across prompts with percentile bootstrap confidence intervals.

Preference win rate. For each prompt, we compare a selected response (system B) against a baseline response (system A) using a panel of judges and evaluate both presentation orders (A/B and B/A). Each judge outcome is mapped back to the system identities and scored as

$$z = \begin{cases} 1 & \text{if } B \text{ is preferred,} \\ 0 & \text{if } A \text{ is preferred,} \\ \frac{1}{2} & \text{if tied.} \end{cases}$$

The per-prompt preference score is the mean judge score across all votes, and the reported Preference is the mean of this score across prompts. We compute 95% confidence intervals using a hierarchical bootstrap that resamples prompts and, within each prompt, resamples judge votes.

Preference margin. We define the tie-aware Margin as a symmetric transformation of Preference:

$$\text{Margin} \;=\; 2\,\text{Preference} - 1,$$

and report it with 95% hierarchical bootstrap confidence intervals (converting to percentage points when reporting in pp).

F Complementary Analysis of Optimization Trajectories

F.1 Mathematical Reasoning

We present complementary optimization trajectory analyses of ANNETRON, GENETRON, MEMETRON, and MEMETRON with correctness shaping for the mathematical reasoning experiment reported in Section 4.1.

To understand the behavior of the optimization process, we analyze the trajectory of Best RM scores across generations, examining marginal gains and improvement across prompt difficulty levels. We approximate prompt difficulty using the Base-16 RM score, where prompts yielding lower Base-16 scores are considered harder, as the base model produces lower-quality responses for these prompts prior to optimization. Figure 4 presents these views for all three methods.

All three methods share a consistent pattern: the largest marginal gain occurs at generation 1, after which improvements diminish rapidly. MEMETRON achieves the highest first-generation gain ($\Delta = 7.75$), followed by GENETRON ($\Delta = 3.83$) and ANNETRON ($\Delta = 2.61$), demonstrating that the integration of GENETRON and ANNETRON within MEMETRON yields coherent and additive optimization gains.

Across all methods, harder prompts benefit more from additional generations. ANNETRON shows Q1 gaining 9.34 points versus 4.45 points for Q4; GENETRON shows Q1 gaining 12.00 points versus 6.55 points for Q4; and MEMETRON shows the largest overall gains, with Q1 improving by 15.70 points and Q4 by 8.30 points. This consistent pattern suggests that MEMETRON and its component optimizers are most beneficial precisely where the base model struggles the most.

Figure 5 presents the optimization trajectory for MEMETRON with correctness shaping. The marginal gain pattern remains similar to vanilla MEMETRON, with a first-generation gain of $\Delta = 7.62$, confirming that the additive correctness bonus does not alter the optimization dynamics. Comparing the generation-level gains,

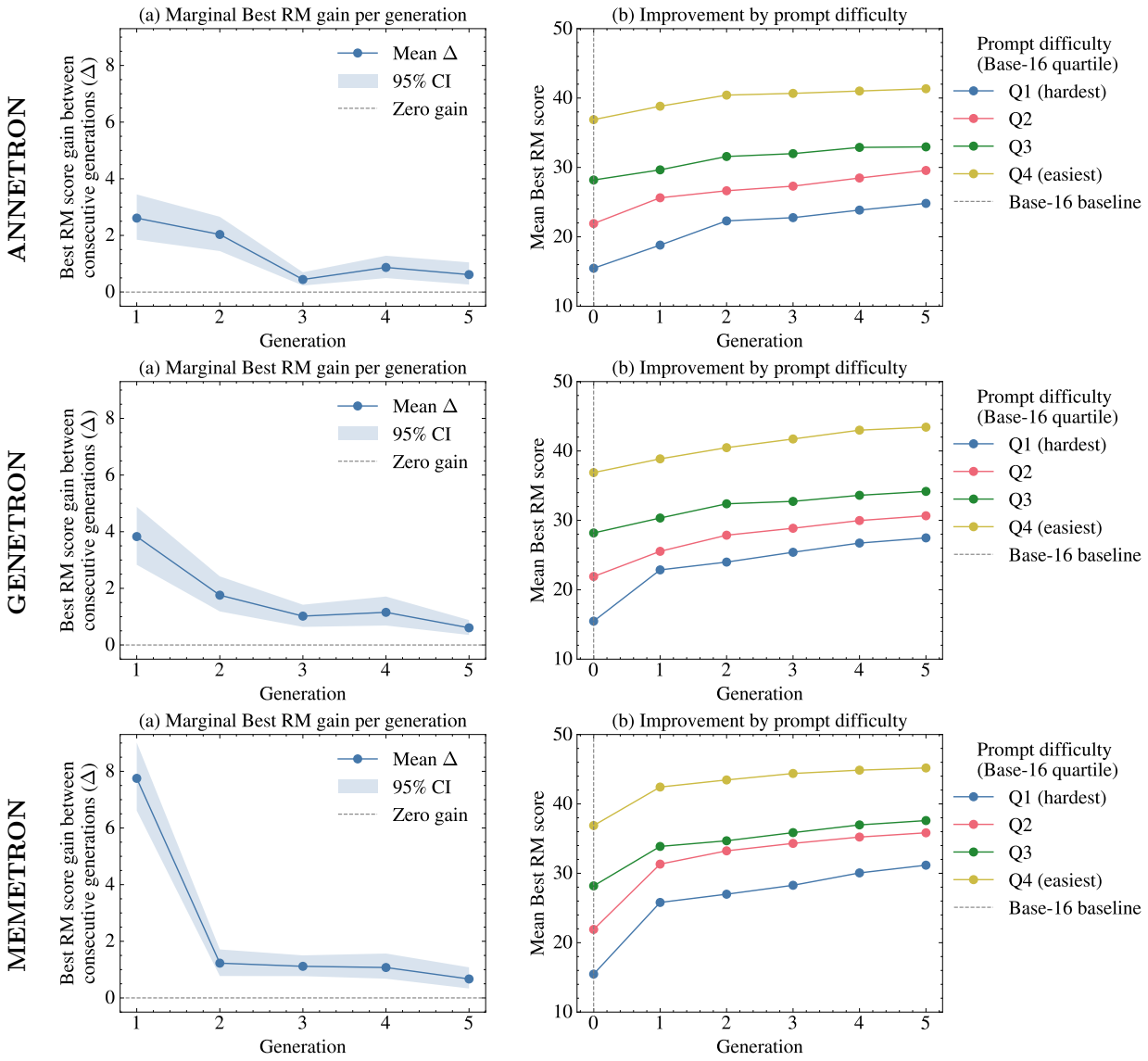

Figure 4: Optimization trajectory across generations for ANNETRON, GENETRON, and MEMETRON on AMC 2025 mathematical reasoning prompts using non-thinking Qwen-3-8B. **(a)** Marginal Best RM score gain between consecutive generations (Δ) with 95% confidence intervals (shaded). Positive values above the zero-gain line (dashed) indicate continued improvement. **(b)** Mean Best RM score across generations stratified by prompt difficulty quartile, defined by Base-16 score. Q1 contains the hardest 25% of prompts and Q4 the easiest. The dashed vertical line marks the Base-16 baseline prior to optimization.

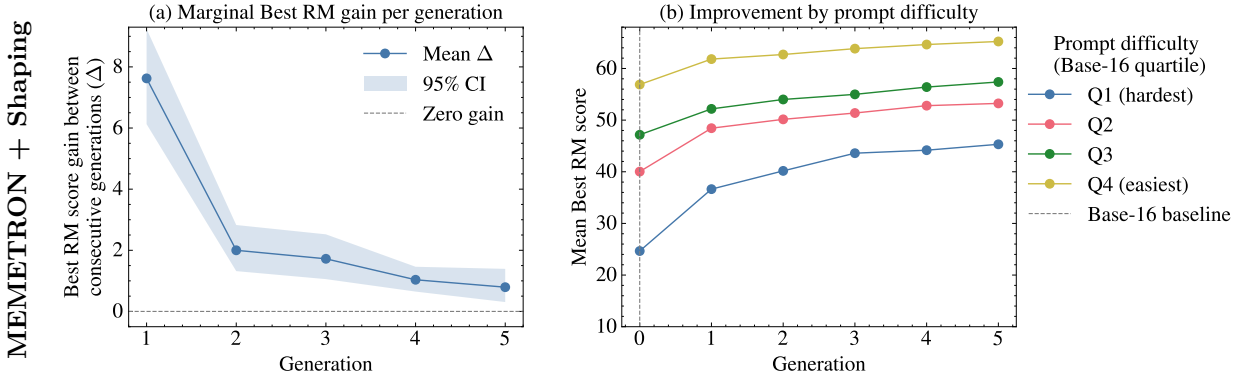

Figure 5: Optimization trajectory across generations for MEMETRON with correctness shaping on AMC 2025 mathematical reasoning prompts using non-thinking Qwen-3-8B. **(a)** Marginal Best RM score gain between consecutive generations (Δ) with 95% confidence intervals (shaded). Positive values above the zero-gain line (dashed) indicate continued improvement. **(b)** Mean Best RM score across generations stratified by prompt difficulty quartile, defined by Base-16 score. Q1 contains the hardest 25% of prompts and Q4 the easiest. The dashed vertical line marks the Base-16 baseline prior to optimization.

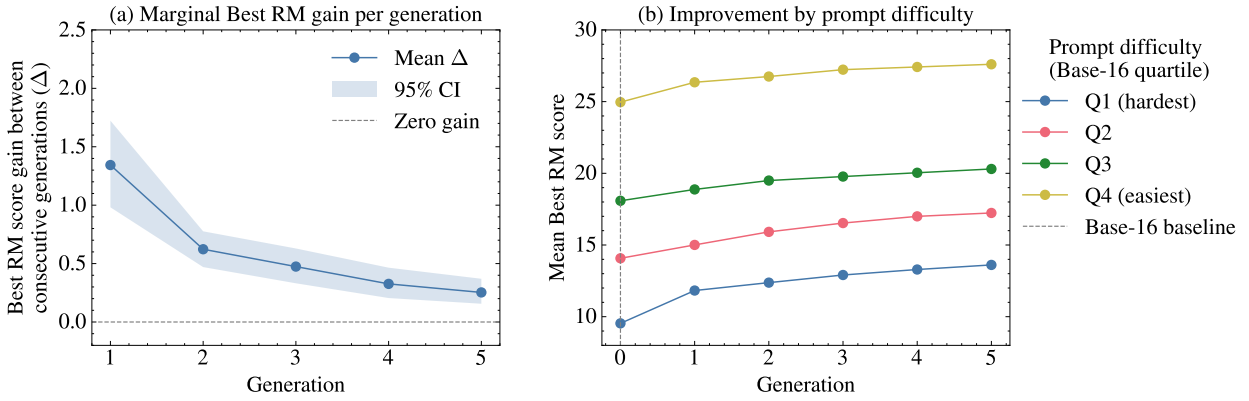

Figure 6: Optimization trajectory across generations for MEMETRON on 100 TinyAlpacaEval Instruction Following prompts using Llama-3.2-3B-Instruct. **(a)** Marginal Best RM score gain between consecutive generations (Δ) with 95% confidence intervals (shaded). Positive values above the zero-gain line (dashed) indicate continued improvement. **(b)** Mean Best RM score across generations stratified by prompt difficulty quartile, defined by Base-16 score. Q1 contains the hardest 25% of prompts and Q4 the easiest. The dashed vertical line marks the Base-16 baseline prior to optimization.

which are invariant to the constant reward shift, reveals that correctness shaping disproportionately benefits harder prompts: Q1 achieves a gain of 20.66 points compared to 15.70 points for vanilla MEMETRON, while Q4 remains largely unchanged (8.34 vs. 8.30 points). This suggests that MEMETRON progressively improves the Best RM score across generations for all difficulty levels, and correctness shaping amplifies this effect for harder prompts by providing an additional reward signal when correct solutions are found.

F.2 Instruction Following

We present complementary optimization trajectory analysis of MEMETRON for the instruction following experiment reported in Section 4.2.

Figure 6 presents the optimization trajectory for MEMETRON on the instruction following task. The pattern is consistent with the mathematical reasoning experiments: the largest marginal gain occurs at generation 1 ($\Delta = 1.34$), after which improvements diminish gradually but remain positive throughout. Across all diffi-

culty levels, the Best RM score improves monotonically across generations, with harder prompts benefiting more overall: Q1 achieves an absolute gain of 4.08 points compared to 2.64 points for Q4, suggesting that MEMETRON is most beneficial where the base model struggles the most, consistent with the findings on mathematical reasoning.

G Implications

MEMETRON provides a unified mechanism for reward-guided optimization over complete LLM responses without modifying model parameters or decoding procedures. Because optimization is formulated purely at the response level and relies only on black-box reward evaluations, the same search process can be deployed both at inference time to improve final outputs and during off-policy training to generate higher-quality training candidates. Moreover, when verifiable correctness signals are available, search-based optimization naturally exposes reward-correctness mismatches, enabling systematic analysis and mitigation of reward misalignment.

G.1 Inference-Time Response Optimization

At inference time, model parameters are fixed, and performance can only be improved by modifying how candidate responses are explored and selected. Rather than relying on independent sampling followed by one-shot selection as in best-of-N and self-consistency, MEMETRON performs structured exploration and iterative refinement of complete responses guided by the reward function, returning

$$\hat{y} = \arg \max_{y \in \mathcal{H}(x)} r(x, y),$$

where $\mathcal{H}(x)$ denotes the set of candidates explored during search. Because optimization operates entirely at the response level and requires only black-box reward evaluations, MEMETRON can be deployed as a post-generation layer on top of any frozen LLM and reward function, without modifying model parameters or token-level decoding procedures.

G.2 Training-Time Response Optimization

MEMETRON can also be used during off-policy training as a response-level search operator that improves how candidate responses are generated. Given a prompt x, a policy π_θ, and a reward function r, MEMETRON produces a set of evaluated candidate responses

$$\mathcal{H}(x) = \text{MEMETRON}(x, \pi_\theta, r),$$

together with their associated reward scores $\{r(x, y)\}_{y \in \mathcal{H}(x)}$. The best-of-search response $\hat{y} = \arg\max_{y \in \mathcal{H}} r(x, y)$ can then be extracted directly from this candidate set.

SFT with reward-optimized targets. Instead of sampling $y \sim \pi_\theta(y \mid x)$ and filtering, we train on the approximate solution $\hat{y}$ produced by MEMETRON, using the standard supervised objective

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{x \sim \mathcal{D}} \log \pi_\theta\big(\hat{y} \mid x\big).$$

This replaces shallow generation-and-filtering with reward-guided search for constructing training targets, with $\hat{y}$ treated as a fixed target during optimization. Since RL pipelines such as PPO and GRPO typically rely on an SFT warmup stage to initialize the policy, MEMETRON can also improve this initialization by supplying higher-reward training targets, leading to a stronger starting point before RL training begins.

DPO with search-induced preference pairs. From the candidate pool $\mathcal{H}(x)$, we construct preference pairs by selecting a high-reward and a low-reward response,

$$y^+ \in \text{Top}_k(\mathcal{H}(x)), \qquad y^- \in \text{Bottom}_k(\mathcal{H}(x)),$$

inducing a preference dataset $\mathcal{P} = \{(x, y^+, y^-)\}$ used in DPO via

$$\mathcal{L}_{\mathrm{DPO}}(\theta) = -\mathbb{E}_{(x,y^+,y^-)\sim\mathcal{P}} \log \sigma\Big(\beta\big(\log \pi_\theta(y^+ \mid x) - \log \pi_\theta(y^- \mid x)\big)\Big).$$

MEMETRON thus supplies informative preferred and dispreferred responses drawn from structured search, while leaving the DPO objective unchanged.

G.3 Reward Misalignment Analysis and Mitigation in Verifiable Tasks

In tasks with verifiable final answers, a learned RM may assign high scores to responses that are incorrect or low-utility, due to limited model capacity, distribution shift, or systematic biases in how surface features are scored. In these cases, reward-guided selection may fail to identify the best candidate even when high-quality solutions are present in the explored set.

We address reward misalignment through a search-driven diagnostic and mitigation workflow centered on correctness shaping, covering three steps: detecting misalignment through coverage-selection discrepancies amplified by search-based optimization, isolating misranking failures using correctness-conditioned reward shaping, and mitigating misalignment via targeted reward model refinement. Crucially, MEMETRON performs search-based discrete optimization rather than independent sampling, which actively concentrates candidates in high-reward regions of the response space, amplifying discrepancies between reward scores and true task outcomes and making misalignment easier to surface and analyze.

Exposing and Detecting Misalignment via Selection Gaps under Search. We detect reward misalignment by comparing coverage-oriented metrics such as Pass@n, which indicate whether correct solutions are present among explored candidates, with selection-based metrics such as best-of-N, which measure whether the RM successfully selects them. When correct solutions exist but are not selected, this indicates reward model misranking. Search-based optimization amplifies such discrepancies by intentionally concentrating candidates in high-reward regions, producing more challenging candidate sets that stress-test the RM and make selection failures easier to quantify.

Correctness Shaping for Isolating Failures and Optimizing Correct Solutions. To isolate reward model misranking and guide search toward correct solutions, we introduce correctness-conditioned reward shaping, which augments the base reward with an explicit correctness signal:

$$r'(x, y) = r(x, y) + c \cdot \mathbb{1}[\text{answer}(y) \text{ is correct}],$$

where $c > 0$ is a constant bonus applied to correct responses. Running search under both the base reward r and the shaped reward r' induces two complementary candidate distributions: responses that achieve high reward under r but are incorrect, and responses that achieve high reward under r' while satisfying correctness constraints. Because both sets are generated using the same search operators, correctness shaping isolates the effect of answer recognition while controlling for search dynamics, and provides a practical oracle-assisted upper reference for achievable selection performance.

Mitigating Reward Misalignment via Search-Generated Preference Data. Correctness shaping naturally yields informative preference pairs (y^+, y^-), where y^+ denotes a correct response selected under r' and y^- denotes an incorrect response that attains high reward under the base reward r. These pairs directly expose cases where the learned RM favors incorrect but highly scored responses.

Such search-generated preference data can be used to refine the RM using standard preference-based objectives such as the Bradley-Terry loss. For a parameterized RM r_ϕ, the loss for a single pair is

$$\mathcal{L}_{\mathrm{BT}}(\phi) = -\log \sigma\big(r_\phi(x, y^+) - r_\phi(x, y^-)\big),$$

which encourages the RM to assign higher scores to correct, high-quality responses than to incorrect but highly scored ones. Once reward model ranking is improved through such updates, the same high-quality responses can be reused for downstream policy optimization using standard alignment pipelines.